`

ABSTRACT

Lung diseases such as Tuberculosis, Pneumonia, and Cancer continue to pose significant public health challenges, where timely and accurate diagnosis is critical for effective treatment and improved patient outcomes. This project introduces an AI-powered web-based application designed to detect and classify lung diseases from chest X-ray images using deep learning techniques.

The system utilizes Convolutional Neural Networks (CNNs) to classify X-rays into four categories: Normal, Tuberculosis, Pneumonia, and Lung Cancer (Nodule). In cases where cancer is detected, a dedicated segmentation module is employed to visually highlight the tumor-affected regions within the X-ray, assisting medical professionals in understanding the severity and localization of the disease.

Developed using Python, Flask, and HTML/CSS, the application features a user-friendly web interface allowing users to upload chest X-ray images and receive real-time predictions. The backend integrates multiple pre-trained models:

- A main classifier for all four disease categories.
- A specialized model for distinguishing between Pneumonia and Tuberculosis.
- A dedicated cancer detection model for identifying lung nodules.

These modular components ensure enhanced prediction accuracy and flexibility.

This tool is intended to assist healthcare professionals by providing rapid, reliable, and interpretable insights, reducing the burden of manual analysis. Future enhancements may include cloud deployment, automated report generation, multilingual support, and integration with hospital information systems, making the application scalable and impactful in real-world clinical environment.

`

CHAPTER 01

INTRODUCTION

1.1 OVERVIEW

In the age of artificial intelligence and rapid technological advancements, healthcare has experienced significant transformation, particularly through the integration of deep learning and computer vision. One of the most impactful applications of these technologies is the automation of disease detection using medical imaging—especially **chest X-rays**, which are widely used for diagnosing pulmonary conditions.

Lung diseases such as Tuberculosis (TB), Pneumonia, and Lung Cancer remain among the leading causes of morbidity and mortality worldwide. Early and accurate diagnosis is critical in improving treatment outcomes and reducing the burden on healthcare systems.

With the growing availability of annotated chest X-ray datasets and advancements in Convolutional Neural Networks (CNNs), deep learning-based diagnostic systems have demonstrated exceptional performance in classifying and localizing abnormalities in lung images. These systems offer radiologists and medical professionals faster, more consistent results, and visual support by highlighting suspicious regions for closer inspection.

This project, titled "Lung Disease Detection using Deep Learning", aims to design and develop a web-based application capable of detecting and classifying four major lung conditions from chest X-ray images:

- Normal (Healthy)
- Tuberculosis (TB)
- Pneumonia
- Lung Cancer (Nodule, Mass)

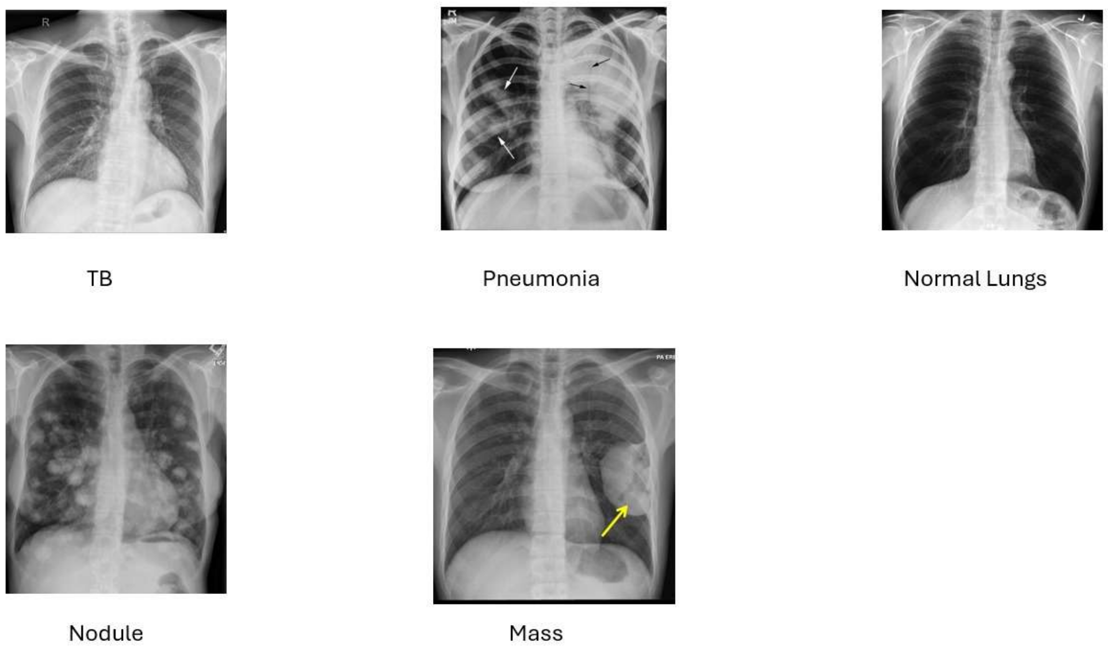


Figure 1.1: Disease Types

The system incorporates three core classification models:

1. **Main Model** – Trained to classify X-rays into all four categories.
2. **Pneumonia vs TB Sub-model** – Focused on improving accuracy between similar conditions.
3. **Cancer (Nodule) Detection Model** – A dedicated classifier for detecting lung nodules indicative of cancer.

WORKING OF MACHINE LEARNING AND DEEP LEARNING MODELS

The process begins with collecting and preparing a dataset of labelled chest X-ray images. Image preprocessing techniques are applied to enhance the features and standardize the input size. This includes grayscale conversion, histogram equalization (CLAHE), noise reduction, and resizing.

A deep learning model, usually a **Convolutional Neural Network (CNN)**, is trained on the dataset. CNNs automatically learn spatial hierarchies of features through layers such as convolution, pooling, and activation. The trained model learns to distinguish between various disease patterns present in the lungs.

Once trained and evaluated, the model is saved and integrated into a **Flask web framework** for deployment. Users can upload X-ray images through the frontend interface, and the backend processes the image, classifies it, and returns the result along with visualizations of the infected areas.

`

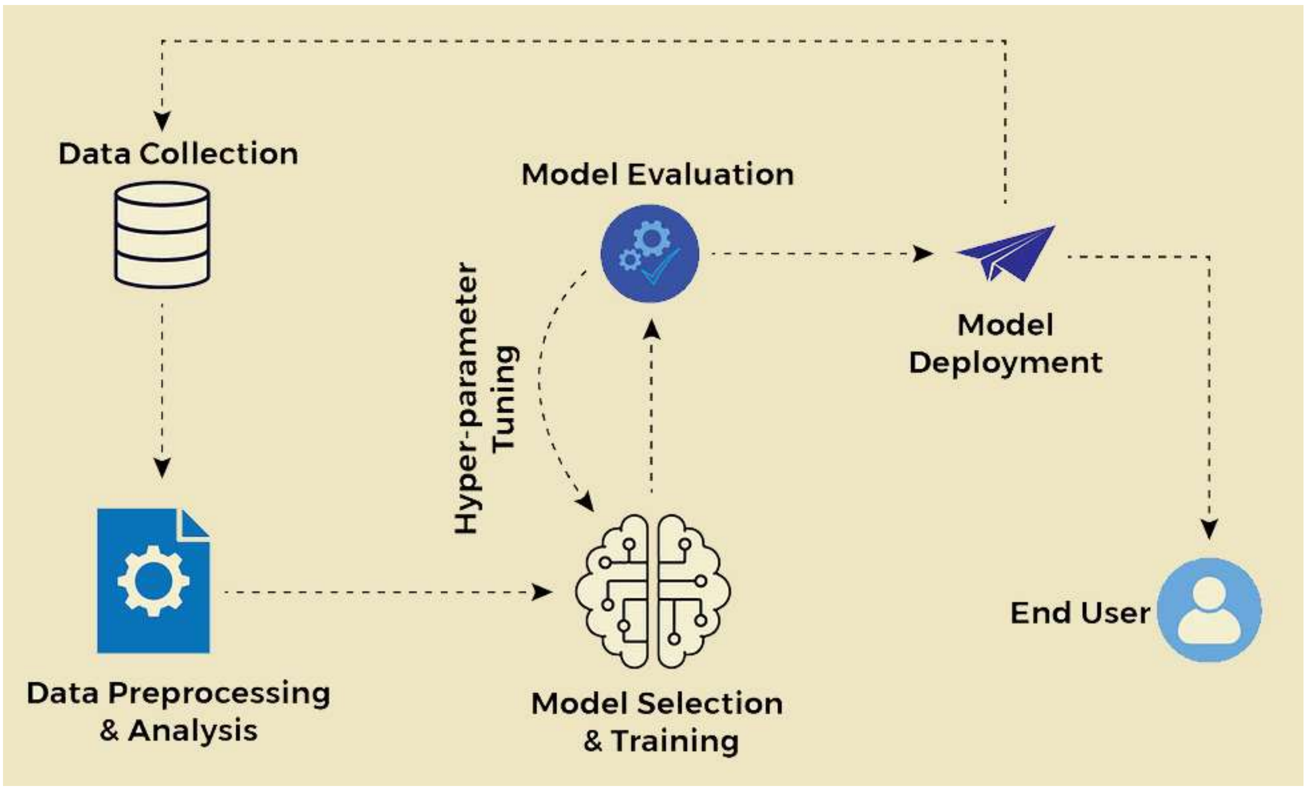


Figure 1.2: Workflow of the Lung Disease Detection System

FEATURES OF DEEP LEARNING-BASED DETECTION SYSTEM

Deep learning is a subset of machine learning that involves neural networks with three or more layers. Here are 12 key features of deep learning:

- **Neural Networks**: Deep learning relies on artificial neural networks, which are inspired by the structure and function of the human brain. These networks consist of interconnected nodes, or neurons, organized into layers.
- **Deep Neural Networks (DNNs):** DNNs have multiple layers (deep architectures), including an input layer, one or more hidden layers, and an output layer. The depth allows the model to learn hierarchical representations of data.
- **Feature learning**: Deep learning algorithms automatically learn hierarchical representations of data. Lower layers capture simple features, and higher layers combine them to form more complex features, enabling the model to understand intricate patterns.
- **Representation Learning**: Deep learning models learn to represent data in a hierarchical manner, extracting features at different levels of abstraction. This facilitates better generalization to new, unseen data.

- **Backpropagation**: Backpropagation is the training algorithm used in deep learning. It involves propagating errors backward through the network, adjusting the weights of connections to minimize the difference between predicted and actual outputs.
- **Activation Functions**: Activation functions introduce non-linearity to neural networks, enabling them to learn complex relationships. Common activation functions include ReLU (Rectified Linear Unit), Sigmoid, and Tanh.
- **Convolutional Neural Networks (CNNs):** CNNs are specialized deep learning architectures designed for image processing. They use convolutional layers to automatically learn spatial hierarchies of features.
- **Recurrent Neural Networks (RNNs):** RNNs are designed for sequential data and have connections that create loops, allowing information persistence. They are widely used in tasks such as natural language processing and time series analysis.
- **Transfer Learning**: Transfer learning involves pre-training a deep learning model on a large dataset and fine-tuning it for a specific task. This approach leverages knowledge gained from one task to improve performance on another.
- **Autoencoders**: Autoencoders are unsupervised learning models that learn efficient representations of data by encoding and decoding it. They are used for tasks such as data compression and feature learning.
- **Dropout**: Dropout is a regularization technique used in deep learning to prevent overfitting. It involves randomly dropping out a fraction of neurons during training, forcing the network to learn more robust features.
- **Generative Adversarial Networks (GANs)**: GANs consist of a generator and a discriminator network that are trained simultaneously. GANs are used for generating new, realistic data samples, making them popular in image generation and other creative applications.

IMPORTANCE OF CNN AND MACHINE LEARNING

Machine learning is crucial for enabling computers to learn from data and make predictions or decisions without explicit programming, while Convolutional Neural Networks (CNNs) are a specific type of machine learning model particularly effective for image and video analysis. CNNs excel at automatically extracting features from images, reducing manual feature engineering and improving efficiency, making them invaluable in computer vision and other related fields.

Here's a more detailed look at their importance:

`

Machine Learning (ML) Importance:

Automation of tasks:

Machine learning algorithms can automate tasks that would otherwise require extensive manual effort, such as image analysis, spam detection, and fraud detection.

Pattern recognition:

ML models can identify patterns in data that humans might miss, leading to better predictions and insights.

Improved accuracy and efficiency:

Well-trained ML models can achieve higher accuracy and efficiency compared to traditional methods, especially in large datasets.

Adaptability and generalization:

ML models can adapt to new data and generalize to unseen situations, making them versatile and useful in various applications.

Personalized experiences:

Machine learning can be used to personalize experiences for users, such as recommending products or content based on their preferences.

Convolutional Neural Networks (CNN) Importance:

Automatic feature extraction:

CNNs automatically learn and extract relevant features from images, reducing the need for manual feature engineering, which is a time-consuming process in traditional image processing.

Spatial invariance:

CNNs are designed to recognize patterns regardless of their position or orientation within an image, making them robust to variations in input.

Efficient image processing:

CNNs are computationally efficient, especially for large image datasets, making them well-suited for real-time applications like autonomous driving and medical imaging.

Strong in computer vision:

CNNs have revolutionized computer vision tasks, such as object detection, image classification, and facial recognition, achieving state-of-the-art accuracy.

Transfer learning:

CNNs can be fine-tuned on new tasks using a small amount of data, making them versatile and efficient for various application

`

1.2 PROBLEM STATEMENT

Lung diseases such as Tuberculosis (TB), Pneumonia, and Lung Cancer are among the most critical public health concerns globally, particularly in rural and underdeveloped regions where access to trained radiologists and diagnostic infrastructure is limited.

While chest X-rays remain a fundamental and accessible diagnostic tool, accurate interpretation requires medical expertise that is often unavailable in such areas. This gap in resources delays diagnosis and treatment, increasing the risk of complications and fatalities.

To address this challenge, we propose an AI-powered web application that automatically classifies chest X-ray images using deep learning models. The system provides:

- Classification of chest X-rays into Normal, TB, Pneumonia, and Lung Cancer categories.
- Visual highlighting of abnormal regions using segmentation techniques, particularly for cancer detection.
- A combined multi-class model and two specialized sub-models for improved accuracy and flexibility.
- A Flask-based user interface for seamless image upload and instant feedback.

This tool aims to assist healthcare providers in early diagnosis and triage, especially in low-resource settings, ultimately helping to reduce diagnostic delays and improve patient outcomes.

Lung diseases are a leading cause of illness and death worldwide, disproportionately affecting populations in rural and resource-constrained regions. Early detection is key, but the scarcity of radiologists and advanced diagnostic tools makes timely diagnosis difficult.

This project develops a deep learning-based web application that enables automated detection of lung diseases from chest X-rays. It offers:

- Automated Classification into:
  - Normal (Healthy)
  - Tuberculosis
  - Pneumonia
  - Lung Cancer (Nodule)
- Visual Infection Highlighting using segmentation, helping identify and understand affected regions.

`

- A user-friendly Flask web interface that allows users—healthcare workers, clinicians, or patients—to upload X-ray images and receive predictions instantly.

The system is tailored for use in rural clinics, community health centers, and emergency setups, where expert analysis is otherwise unavailable.

1.3 MOTIVATION

Millions of individuals across the globe suffer due to delayed or missed diagnoses of lung diseases, especially in areas lacking healthcare infrastructure.

Key Motivators:

- Improving Rural Healthcare Access:
  - Primary healthcare centers in rural areas are often ill-equipped.
  - An AI-based tool offers a first line of screening that supports both doctors and patients.
- Reducing Diagnostic Dependency:
  - Radiologists are in short supply in many regions.
  - An automated system can:
    - Provide preliminary analysis
    - Prioritize critical cases
    - Support non-specialist medical personnel with AI-driven insights
- Addressing Global Health Gaps:
  - According to the World Health Organization:
    - TB continues to affect vulnerable populations worldwide.
    - Pneumonia is a leading cause of death in children under five.
    - Lung cancer has high mortality rates due to late detection.

This project aims to empower under-resourced healthcare environments with reliable diagnostic assistance powered by AI.

Vision

To bridge the gap between urban and rural healthcare by delivering a scalable, AI-driven diagnostic platform that is:

`

- **Accessible** – Requiring only basic internet and device access.
- **Affordable** – Built with open-source tools to minimize cost.
- **Effective** – Using deep learning for high accuracy.
- **User-Friendly** – With a clean web interface for ease of use.

Ultimately, the goal is to make expert-level diagnostic support available to everyone, anywhere—supporting early detection, timely treatment, and better health outcomes.

INTRODUCTION TO LUNG DISEASE DETECTION

Lung diseases such as Tuberculosis (TB), Pneumonia, and Lung Nodules are among the most serious global health concerns, particularly in rural and under-resourced regions with limited access to medical specialists. Early and accurate diagnosis plays a critical role in reducing mortality rates, but traditional diagnostic methods like manual X-ray interpretation require experienced radiologists and are often time-consuming and subjective.

To address this challenge, this project proposes a deep learning-based solution for automated lung disease classification using chest X-ray images. By utilizing Convolutional Neural Networks (CNNs) trained on labeled medical datasets, the system is capable of distinguishing between four major conditions: Normal, Tuberculosis (TB), Pneumonia, and Lung Nodules.

The aim of the project is to design a lightweight, accessible, and accurate diagnostic aid that can assist doctors, healthcare workers, and even patients in remote or underserved areas. Through image preprocessing, model training, and web-based deployment, the system provides instant predictions for uploaded X-ray images, thereby supporting faster diagnosis and better treatment planning.

A key strength of the proposed system is its ability to deliver real-time classification through a Flask-based web application, which integrates a simple user interface where users can upload an X-ray and receive both a predicted diagnosis and a confidence score. The system can be extended or fine-tuned to support additional diseases or integrated with hospital record systems.

In summary, this project combines the power of Artificial Intelligence with medical imaging to build a scalable solution for automatic lung disease detection that is especially beneficial for low-resource healthcare settings.

1.4 OBJECTIVE:

The primary goal of this project is to provide an accessible, accurate, and efficient diagnostic support tool to identify lung-related diseases — including Tuberculosis (TB), Pneumonia, Lung Nodules, and Normal cases — using chest X-ray images. By applying Convolutional Neural Networks (CNNs) and

`
deploying them through a simple Flask web interface, this system empowers doctors and healthcare providers, especially in rural areas, to detect diseases early and reduce misdiagnosis.

KEY COMPONENTS:

- **CNN-Based Deep Learning Model:** The core component is a trained deep learning model built with CNN architecture. The model learns features such as textures, patterns, and densities from X-ray images to classify them into disease categories.
- **Web Application (Flask):** The model is deployed in a web environment using Flask, allowing users to upload X-ray images and get real-time predictions. This increases usability even in non-technical healthcare environments.
- **Preprocessing Pipeline:** Before making predictions, uploaded images are resized, normalized, and processed to match the input requirements of the model.

BENEFITS:

- **Accessibility:** Can be used in rural and low-resource settings via a simple web interface.
- **Accuracy:** Deep learning improves detection performance over traditional rule-based methods.
- **Speed:** Instant predictions reduce diagnosis time, especially during medical emergencies.
- **Scalability:** Easily extendable to other diseases with retraining on additional datasets.

1.5 CHALLENGES AND CONSIDERATIONS:

- **Dataset Imbalance:** Unequal representation of disease categories may lead to biased predictions.
- **Explainability:** While CNNs are powerful, understanding "why" they make a prediction remains a challenge.
- **Real-world Variability:** Differences in X-ray quality, position, or format may affect prediction accuracy.
- **Validation:** Continuous validation and fine-tuning are necessary for medical-grade reliability.

1.6 CONCLUSION:

The implementation of this **AI-driven lung disease detection system** signifies a powerful step toward accessible healthcare. By combining deep learning with easy-to-use web deployment, it democratizes diagnostic support and helps bridge the gap in medical facilities in underserved areas. Future enhancements could include expanding disease classes, integrating with hospital records, or mobile-based deployment to reach even more users.

`

CHAPTER 02

LITERATURE SURVEY

Lung disease detection using chest X-ray has become one of the most active areas of research in the field of medical imaging, especially with the integration of deep learning techniques. Convolutional Neural Networks (CNNs) and other advanced deep learning architectures have enabled automatic detection of various lung conditions such as tuberculosis, pneumonia, and lung cancer. Several researchers have proposed models with improved accuracy and reliability, aimed particularly at resource-constrained environments where expert radiologists are not always available.

Early approaches used traditional image processing and machine learning techniques, which required handcrafted features and manual tuning. With the rise of deep learning, researchers began using pretrained networks and large-scale datasets to train models capable of outperforming traditional techniques. The use of transfer learning, data augmentation, and segmentation-based methods has further improved the accuracy and interpretability of these systems. Many of these studies have focused on creating scalable, low-cost, and reliable tools for aiding doctors, especially in rural or underdeveloped areas.
The table below summarizes a few important research contributions in this domain:

Table 2.1: Survey of Research on Lung Disease Detection Using Deep Learning

Title	**Author(s)**	**Year**	**Approach**
ChestX-ray8: Hospital-Scale Chest X-ray Database	Wang et al.	2017	Released a large-scale dataset with over 100k chest X-rays and used a CNN for detecting 14 thoracic diseases.
CheXNet: Radiologist-Level Pneumonia Detection	Rajpurkar et al.	2017	Introduced DenseNet-121 trained on ChestX-ray14 dataset to detect pneumonia with performance comparable to radiologists.
Deep Learning for Tuberculosis Screening	Lakhani & Sundaram	2017	Applied pretrained CNN models (AlexNet, GoogLeNet) to detect TB in chest radiographs with high accuracy.
3D CNN for Lung Cancer Prediction	Ardila et al. (Google)	2019	Used 3D CNN for lung cancer screening from CT scans, outperforming radiologists in some cases.
Deep Learning-Based	Stephen et al.	2020	Built a CNN-based model to classify lung diseases from chest

`

Title	Author(s)	Year	Approach
Classification of Lung Diseases			X-rays with segmentation overlays for interpretability.
Transfer Learning for Chest Disease Detection	Islam et al.	2020	Employed pretrained models like ResNet and VGG16 for detecting TB and pneumonia in small datasets.
Hybrid CNN-RNN for Pneumonia Detection	Liang & Zheng	2021	Proposed a hybrid model combining CNN and RNN to better classify pneumonia based on sequential X-ray features.

`

CHAPTER 03

SYSTEM ANALYSIS AND DESIGN

INTRODUCTION

This chapter outlines the system analysis and design methodology for the project titled "Lung Disease Detection using Deep Learning and X-rays." The goal is to develop an intelligent system that assists medical practitioners in diagnosing lung-related conditions such as Tuberculosis (TB), Pneumonia, Lung Nodules, and Normal cases using chest X-ray images.

Given the increasing burden on healthcare systems, especially in rural and under-resourced regions, there is a pressing need for fast, accurate, and accessible diagnostic tools. Manual inspection of chest X-rays is both time-consuming and requires specialized expertise. This project introduces a solution by automating diagnosis using Convolutional Neural Networks (CNNs) deployed via a simple web application.

The application accepts X-ray images through a Flask-based web interface, processes them using a trained CNN model, and predicts the presence of lung diseases with high accuracy. Additionally, the project incorporates Grad-CAM visualization to highlight regions in the lungs that influenced the model's prediction—improving the interpretability of AI decisions.

The following sections describe the software requirements, functional/non-functional features, and design architecture, which ensure the modularity, reliability, and effectiveness of the system.

3.1 SOFTWARE REQUIREMENTS

Development Tools

- **Google Colab/ Jupyter Notebook**:
  Used for training and evaluating deep learning models with GPU support. Facilitates rapid prototyping and experimentation on cloud or local notebooks.
- **PyCharm**:
  Integrated Development Environment (IDE) for developing the Flask backend and organizing the project structure for local deployment.
- **Flask**:
  A lightweight Python web framework used to serve predictions via REST APIs and render the web interface for model inference.

`

Python Libraries Used

- **OpenCV**:
  For image processing operations like resizing, grayscale conversion, and enhancement using CLAHE.
- **NumPy**:
  For numerical operations and array manipulation during preprocessing and model input formatting.
- **TensorFlow & Keras**:
  For building, training, and deploying Convolutional Neural Network (CNN) models for image classification.
- **Matplotlib / Seaborn**:
  Used for visualizing accuracy, loss, and confusion matrix during training and testing phases.
- **Werkzeug**:
  Utility library used by Flask for secure file uploads and path management.
- **OS / UUID**:
  Used to manage file directories, file paths, and create unique file names for uploaded images.

Deep Learning Components (Keras Layers)

- **Conv2D**:
  Performs 2D convolution operations to extract spatial features from X-ray images.
- **MaxPooling2D**:
  Reduces dimensionality of feature maps and helps in focusing on the most prominent features.
- **Flatten**:
  Converts the 2D matrix from convolutional layers into a 1D vector for fully connected layers.
- **Dense**:
  Fully connected layers used at the end of the network to make final predictions based on learned features.
- **Dropout**:
  Regularization technique to prevent overfitting by randomly turning off a fraction of neurons during training.
- **Sequential CNN**:
  Model architecture builder used to stack layers linearly in Keras.

`

3.2 SOFTWARE REQUIREMENT SPECIFICATION DOCUMENT (SRS)

Functional Requirements

1. **Image Upload and Preprocessing**
    - Users should be able to upload chest X-ray images via the web interface.
    - Images will be converted to grayscale, resized (256×256), and normalized.
2. **Disease Prediction using CNN**
    - The system uses a trained CNN model to classify images into one of four classes:
        - **Normal**
        - **Tuberculosis**
        - **Pneumonia**
        - **Nodule**
    - Prediction confidence (in %) will also be displayed.
3. **Web Interface (Flask)**
    - Provides:
        - Homepage
        - Image upload section
        - Result display (disease, confidence score, and heatmap)
        - Navigation to individual models (TB/Pneumonia, Lung Cancer)
4. **Image Storage and Output**
    - Uploaded images and prediction outputs are stored in a static/uploads/ folder.
    - Prediction results and visual overlays are rendered via HTML templates (result.html).

Non-Functional Requirements

- **Reliability:**
    - The system must work consistently with minimal downtime.
    - Robust error handling for file uploads and missing image data.
- **Usability:**
    - Simple, clean UI suitable for healthcare workers and non-technical users.
    - No login or account required to make predictions.
- **Performance:**
    - The system should respond within **2–3 seconds** after image upload.
    - Optimized CNN inference and image preprocessing for quick predictions.

`

- **Security:**
  - Uploaded files are sandboxed and deleted regularly (optional enhancement).
  - No user-identifiable data is stored.
- **Scalability:**
  - Modular structure allows adding more disease classes or switching to transfer learning models like ResNet/EfficientNet in the future.

3.3 DESIGN APPROACH

1. Modular Architecture

The system follows a **layered architecture** with separate modules for:

Layer	**Responsibility**
Data Layer	Upload and store input X-ray images
Model Layer	Load trained CNN models and perform predictions
Visualization	Generate Grad-CAM heatmaps
Presentation	Display results and heatmaps via HTML pages

2. Data Preprocessing Pipeline

Before predictions, X-ray images undergo preprocessing:

- Grayscale conversion
- Resizing to 256×256
- Normalization (pixel scaling 0–1)
- Batch and channel dimension addition

Libraries used: OpenCV, NumPy, TensorFlow/Keras

3. CNN-Based Classification Model

- A **Convolutional Neural Network** is trained using labeled chest X-rays.
- Output layer uses Softmax activation for multiclass classification.
- Final prediction is determined by the class with highest probability.
- Class labels: ['Normal', 'TB', 'Pneumonia', 'Nodule']

.

4. Flask Web Application

Built using Flask, the web app allows:

- Navigation between model modules

`

- Image upload and preview
- Prediction display with Grad-CAM overlay
- File storage and organization in the static folder

HTML templates (index.html, result.html) are used for frontend rendering.

5. Scalability and Future Extensions

- Models can be swapped (e.g., from CNN to ResNet) without affecting the frontend.
- Additional modules (e.g., COVID-19 detection) can be added easily.
- Mobile-friendly version and deployment via Render or Hugging Face Spaces possible.

.

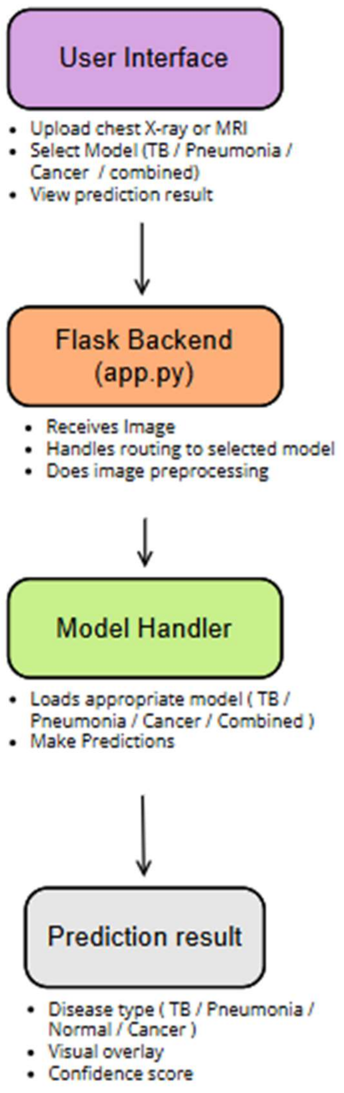

User Interface
• Upload chest X-ray or MRI
• Select Model (TB / Pneumonia / Cancer / combined)
• View prediction result
Flask Backend (app.py)
• Receives Image
• Handles routing to selected model
• Does image preprocessing
Model Handler
• Loads appropriate model (TB / Pneumonia / Cancer / Combined)
• Make Predictions
Prediction result
• Disease type (TB / Pneumonia / Normal / Cancer)
• Visual overlay
• Confidence score

Figure 3.1: System Architecture

`

3.4 UML DIAGRAMS

UML (Unified Modelling Language) is a visual modeling language used in software development to represent the system's design, structure, and behavior. It consists of a set of diagrams such as use case, class, sequence, activity, and state machine diagrams that aid in communication among stakeholders. UML is a standard language used in the software industry, enabling developers to create high-quality, maintainable, and scalable software systems. UML is flexible and can be adapted to various software development processes, making it suitable for different types of projects. Overall, UML is a powerful tool that helps in designing, documenting, and communicating software systems effectively.

An activity diagram is a type of UML diagram used to model the flow of actions and activities within a system or process. It shows the sequence of activities, actions, and decisions that occur from the start to the end of a process or workflow. The diagram consists of nodes and edges that represent activities and transitions between activities. Activity diagrams are useful for analyzing, documenting, and designing complex systems or processes, and they provide a clear and easy-to-understand visual representation of the system's behavior. They are often used in software engineering to model the behavior of software systems and to aid in the design and implementation of business processes.

A class diagram is a type of UML diagram used to model the structure of object oriented systems. It shows the classes in a system, their attributes, methods, and the relationships between the classes. A class is represented as a rectangular box with the class name, and its attributes and methods are listed within the box. The relationships between classes are represented as lines between the classes, showing the type of relationships such as association, aggregation, or inheritance. Class diagrams are useful for designing, analyzing, and communicating the structure of a system, as well as for generating code from the diagram. Class diagrams can also be used for 16 modeling databases, where the classes represent tables, and the relationships represent the foreign key constraints

A sequence diagram is a UML diagram that shows the interactions between objects or components in a system over time. It uses lifelines to represent objects and messages to represent communication between them, including parameters and return values. Sequence diagrams are useful for analyzing, designing, testing, and debugging complex systems. They provide a detailed view of the system's behavior, helping developers to identify and resolve issues.

A state diagram is a type of UML diagram that shows the states and transitions of an object or system. It represents the behavior of the system as a finite state machine, showing how the system transitions from one state to another based on events or conditions. States are represented as rectangles with the state

`

name, while transitions are represented as arrows showing the direction of the transition and the event or condition triggering it. State diagrams are useful for modeling and analyzing the behavior of complex systems, such as software systems or business processes.

3.4.1 Use Case diagram:

The Use Case Diagram illustrates the interactions between various actors (users) and the system, showing what functionalities each actor can perform. It helps to visualize the system's functionality from a user's perspective and identify how different users interact with the system.

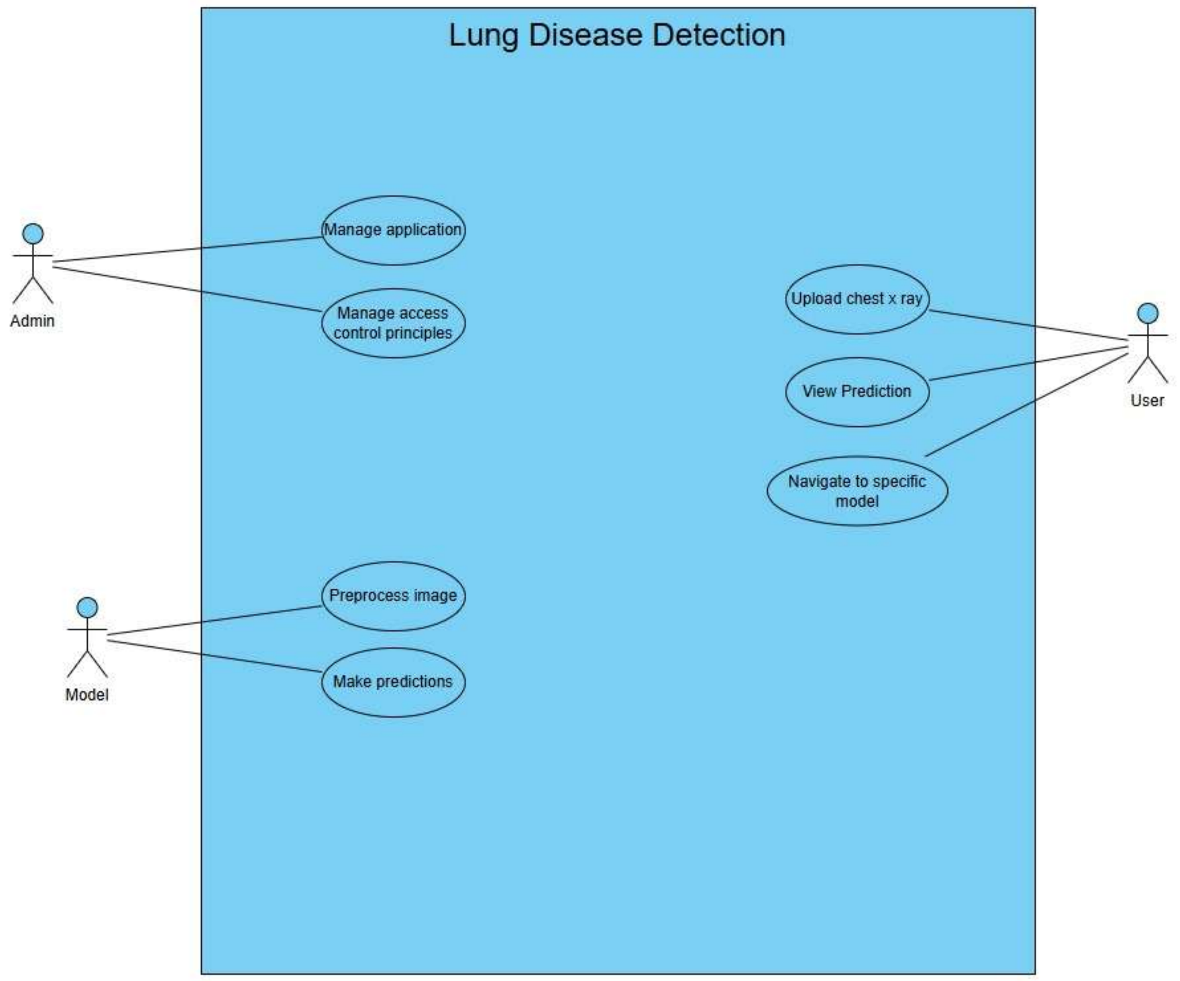


Figure 3.2: Use Case Diagram

Actors Involved:

1. **User**:
    - Uploads chest X-ray images to the system.
    - Views the prediction output.
    - Navigates to specific models (e.g., TB, Pneumonia, Cancer, etc.) for more precise analysis.

`

2. **Admin**:
    - Manages the overall application.
    - Controls access permissions and manages user roles if required.
3. **Model** (as a system component acting like an automated agent):
    - Preprocesses input chest X-ray images.
    - Makes predictions using the trained machine learning/deep learning models.

Use Cases:

- **Upload Chest X-ray**: The user uploads an image via the user interface.
- **View Prediction**: After processing, the user can see the predicted result.
- **Navigate to Specific Model**: Users may choose a specialized model (e.g., only TB or cancer detection) for more focused results.
- **Preprocess Image**: The model component applies preprocessing techniques such as resizing, normalization, and noise removal.
- **Make Predictions**: The model performs inference to classify the disease based on the input image.
- **Manage Application**: The admin oversees the functioning of the system including updates and performance.
- **Manage Access Control Principles**: The admin controls who can access or modify different parts of the system.

3.4.2 Class diagram:

The class diagram of the AI-powered Lung Disease Detection System is designed to showcase a modular, maintainable architecture that adheres to object-oriented design principles and the separation of concerns. Each class in the system is responsible for a specific function—ranging from handling user inputs to producing AI-generated predictions and visual outputs. This design enables smooth data flow from image upload to disease classification and visualization.

The class diagram outlines a scalable structure that supports multiple disease classification models and visualization modules:

`

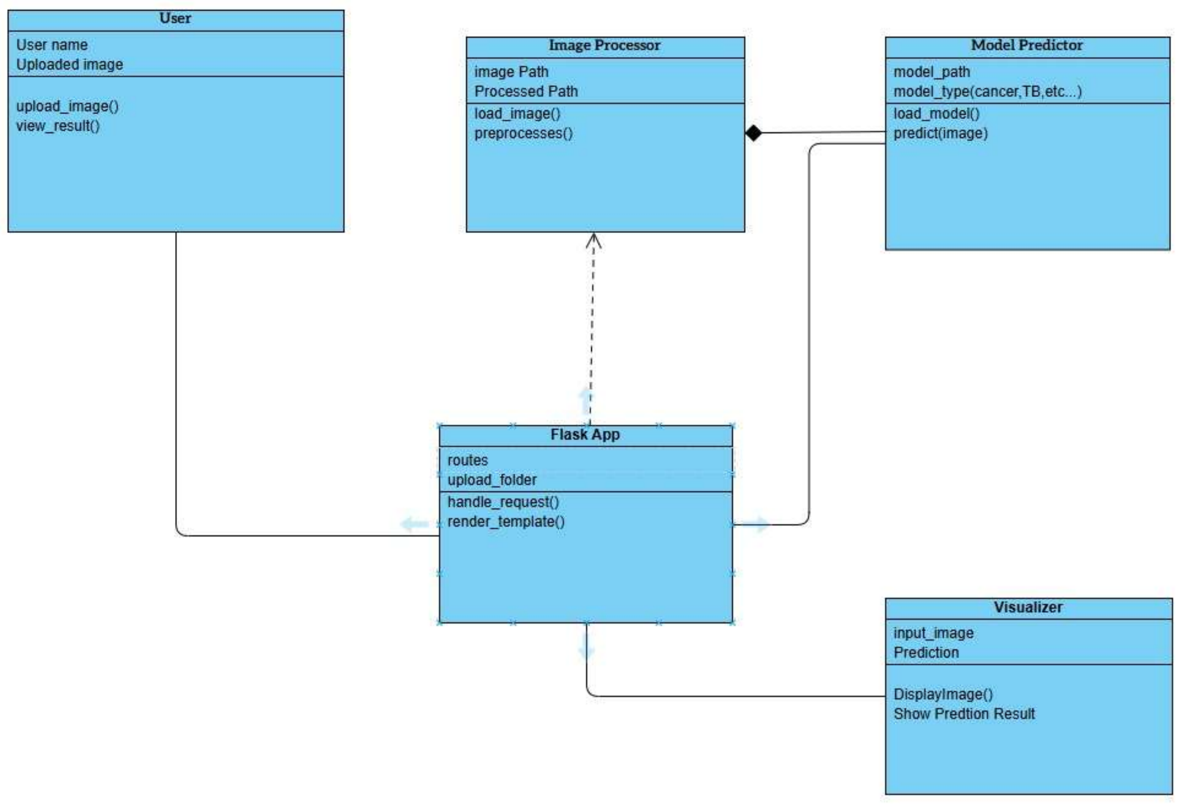


Figure 3.3: Class Diagram

- **User Interface Handler**
  Captures and processes user inputs, primarily the X-ray image file. It also routes the input to the selected model. This serves as the entry point of the system.
- **Image Manager**
  Handles image upload, validation, resizing, and conversion. Prepares the image in the correct format (e.g., grayscale, resized to model input shape) for analysis.
- **Model Selector**
  Allows users to choose between different disease-specific models, such as Cancer, TB, Pneumonia, or a Combined model. Acts as the interface between frontend input and backend model loading.
- **Disease Model (Classifier)**
  Contains trained deep learning models for disease detection. Processes the image and returns the predicted class (e.g., TB, Normal, Cancer, Pneumonia) along with confidence scores.
- **Visualization Module**
  Applies techniques like CLAHE (Contrast Limited Adaptive Histogram Equalization) or CNN-based segmentation to highlight infected regions in the image, improving interpretability of results.

`

- **Prediction Result Handler**
  Stores and formats the model output. Displays predictions, confidence percentages, and segmented images on the results page.
- **Admin Panel (Optional)**
  For system monitoring, model updates, and usage statistics tracking (included for future scalability).
- **Navigation Controller**
  Manages routing between different pages like home, cancer model, TB/Pneumonia model, lung cancer model, and the combined model.

3.4.3 Sequence diagram:

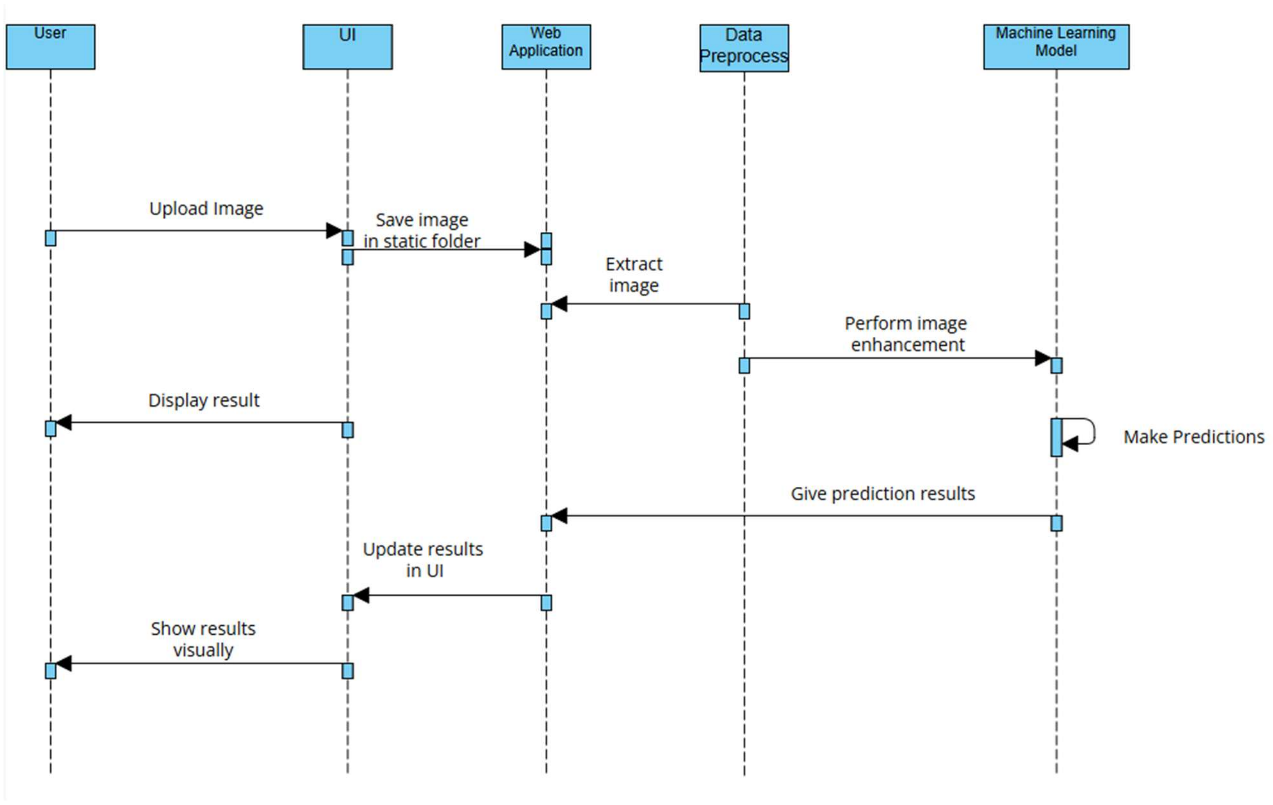


Figure 3.4: Sequence Diagram

This section illustrates the dynamic behavior and interaction between different modules involved in the AI-based Lung Disease Detection System. The sequence diagram outlines the flow of activities starting from user interaction through the frontend UI to the backend processing pipeline, which includes image preprocessing and machine learning inference. It visualizes how the system responds to an X-ray image upload, processes it, and provides prediction results.

`

3.4.4 Activity diagram:

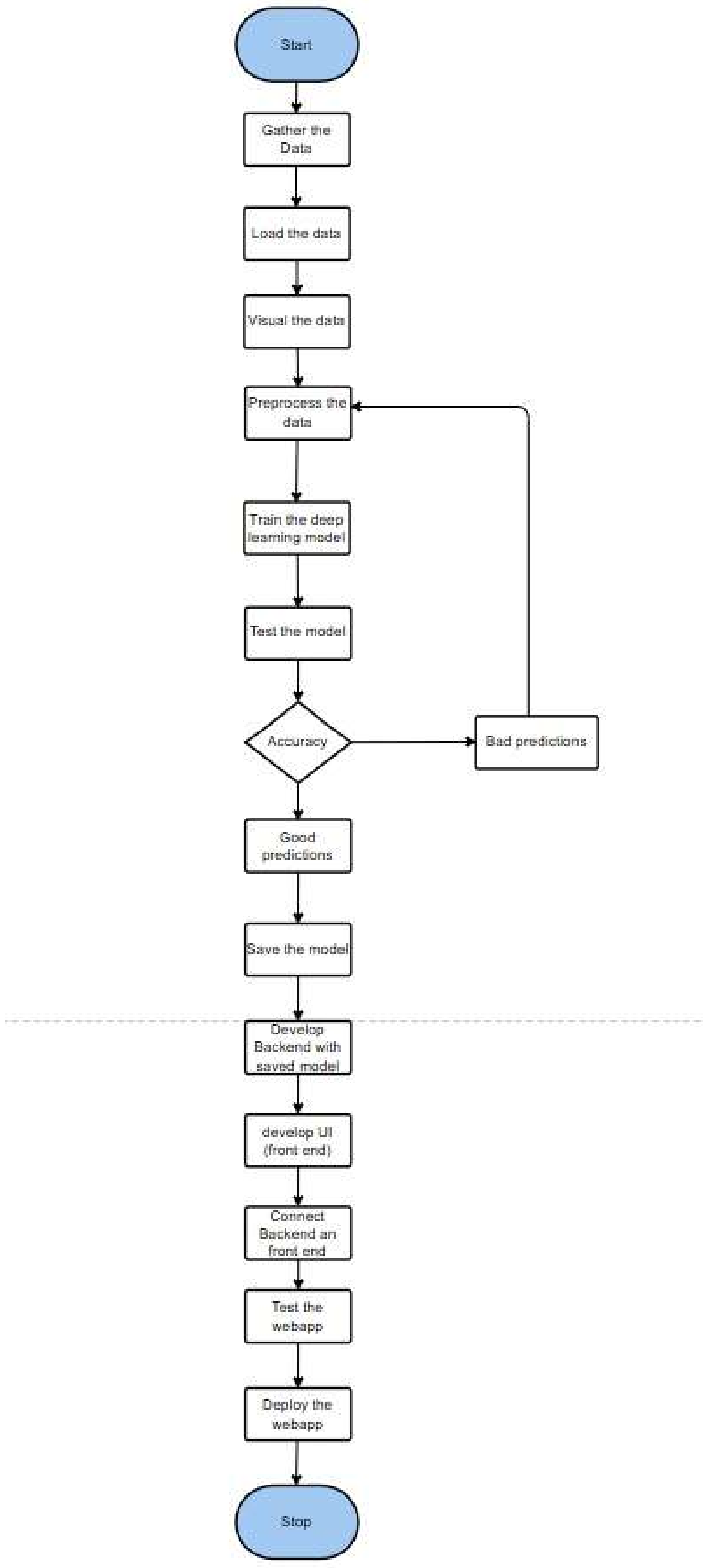


Figure 3.5: Activity Diagram

`

1. Start

The process begins when the system development is initiated.

2. Gather the Data

This step involves collecting chest X-ray images from reliable datasets for diseases like TB, Pneumonia, Lung Cancer, and Normal cases.

3. Load the Data

The collected data is loaded into the development environment for further processing.

4. Visualize the Data

Here, the data is visually analyzed to understand the distribution and quality of images, detect any imbalances, and identify abnormalities.

5. Preprocess the Data

Images are cleaned, resized, normalized, and augmented if necessary to make them suitable for model training. If the model underperforms later, this step may need to be revisited.

6. Train the Deep Learning Model

The system is trained using Convolutional Neural Networks (CNNs) or other deep learning architectures for classification tasks.

7. Test the Model

The model is evaluated using test data to verify its performance.

8. Accuracy Check (Decision Node)

A decision is made based on the model's prediction accuracy:

- If predictions are bad, return to the Preprocess the Data step for improvements.
- If predictions are good, continue with saving and deployment.

9. Save the Model

The trained and validated model is saved for later integration with the web application.

10. Develop Backend with Saved Model

`

A Flask backend is developed where the saved model is integrated to handle image prediction.

11. Develop UI (Frontend)
A user interface is created to allow users to upload images and view prediction results easily.

12. Connect Backend and Frontend
APIs are created to connect the frontend with the Flask backend to enable communication and functionality.

13. Test the Web App
The complete system (frontend + backend) is tested for usability, functionality, and accuracy.

14. Deploy the Web App
Once fully tested, the web application is deployed to a cloud platform for public access.

15. Stop
The system is now live and ready for user interaction.

This section illustrates the dynamic behavior and interaction between different modules involved in the **Lung Disease Detection System**. The sequence diagram outlines the flow of activities starting from user interaction through the frontend UI to the backend processing pipeline, which includes image preprocessing and machine learning inference. It visualizes how the system responds to an X-ray image upload, processes it, and provides prediction results.

3.5 METHODOLOGY AND ALGORITHM

3.5.1 Methodology Overview

The proposed system utilizes a deep learning-based methodology for the automated detection and classification of lung diseases using medical imaging, specifically chest X-rays and lung MRIs. It integrates image preprocessing, multi-stage classification, and intelligent visualization within a user-friendly web application to assist healthcare professionals in diagnosis.

The approach involves multiple deep learning models trained on labelled medical imaging datasets, designed to differentiate between major lung conditions such as **Tuberculosis (TB), Pneumonia, Lung Cancer**, and **Normal (healthy)** lungs. The system also provides segmentation-based infection highlighting to improve interpretability, particularly in cancer detection.

The entire pipeline is broken down into several critical stages that transform raw X-ray data into accurate, actionable diagnostic predictions.

`

The system is composed of five key modules:

1. Image Preprocessing
2. Model Training
3. Flask Web Interface
4. Model Loader & Selector
5. Model Prediction
6. Result Display

Stage 1: Image Data Acquisition

The process begins with a user uploading a chest X-ray scan through the web application. The image can belong to any of the supported categories: TB, Pneumonia, Lung Cancer, Normal, etc. The data includes:

- Medical image file (X-ray)
- TB X-ray
- Pneumonia X-ray
- Nodule / Mass X-ray

The system supports standard image formats (JPG, PNG) and checks for valid file types before proceeding to preprocessing.

Stage 2: Image Preprocessing and Enhancement

Once an image is uploaded, it undergoes a series of preprocessing operations to standardize and enhance image quality for accurate model input:

- Grayscale Conversion: Reduces dimensionality and focuses on intensity-based features.
- Resizing: All images are resized to a consistent dimension suitable for the CNN input (e.g., 224x224).
- Normalization: Pixel values are normalized to scale inputs between 0 and 1.
- CLAHE (Contrast Limited Adaptive Histogram Equalization): Enhances local contrast to improve visibility of structural abnormalities.
- Augmentation (during training): Applies transformations like rotation, flipping, and zoom to increase dataset diversity and prevent overfitting.

This step ensures that all input data is clean, consistent, and optimized for deep learning model inference.

Stage 3: Multi-Stage Deep Learning Classification

The system leverages a hierarchical classification approach involving multiple models:

- **Main Classifier:** A CNN model trained to classify images into four categories:
  - Normal

`

    - Tuberculosis (TB)
    - Pneumonia
    - Nodule (potential cancer)

- **Sub-Model 1 (TB vs Pneumonia):** Invoked when the main model predicts either TB or Pneumonia. It offers more refined discrimination between these similar conditions.
- **Sub-Model 2 (Cancer Detection):** Specialized for classifying between:
    - Normal
    - Nodule Chest X-ray

This architecture enables both broad detection and fine-grained diagnosis.

Stage 4: Prediction Output and Visualization

Once a prediction is generated, the system displays:

- **Predicted Disease Category**
- **Model Confidence Score**
- **Advice/Alert Message:**
  For critical cases (e.g., high likelihood of cancer or TB), a warning or urgent consultation message is displayed.

The outputs are rendered on a result page with a clean and informative interface.

Stage 5: Model Evaluation and Accuracy Validation

To ensure the reliability of the models, performance is evaluated using test datasets with labelled ground truth:

- **Accuracy, Precision, Recall, F1-Score:** Measured for each class.
- **Confusion Matrix:** Analysed to understand inter-class misclassifications.
- **ROC-AUC (for binary models):** Evaluates discriminative ability of sub-models.

Misclassified or low-confidence cases are flagged for retraining or model refinement.

Stage 6: Web Application Integration and Deployment

The trained models are saved in HDF5 or TensorFlow format and integrated into a Flask-based backend:

- **Frontend:** HTML/CSS with Flask templating for upload and result display.
- **Backend:** Handles model loading, image preprocessing, inference, and output formatting.
- **Testing:** Ensures full-stack functionality across different devices and browsers.
- **Deployment:** The app is deployed on cloud platforms such as Heroku or AWS for remote access.

`

This makes the diagnostic system accessible and scalable, even in low-resource settings.

Algorithm Summary

The high-level algorithm can be described as follows:

1. Accept medical image input from the user.
2. Preprocess the image: grayscale, resize, normalize, and enhance.
3. Pass the image through the main classification model.
4. If result is TB or Pneumonia, use Sub-Model 1 for detailed classification.
5. If result is Nodule, use Sub-Model 2 for cancer-related classification.
6. Generate and display prediction with confidence score.
7. If applicable, highlight infected regions via segmentation.
8. Log the prediction and result for potential future retraining or audit.

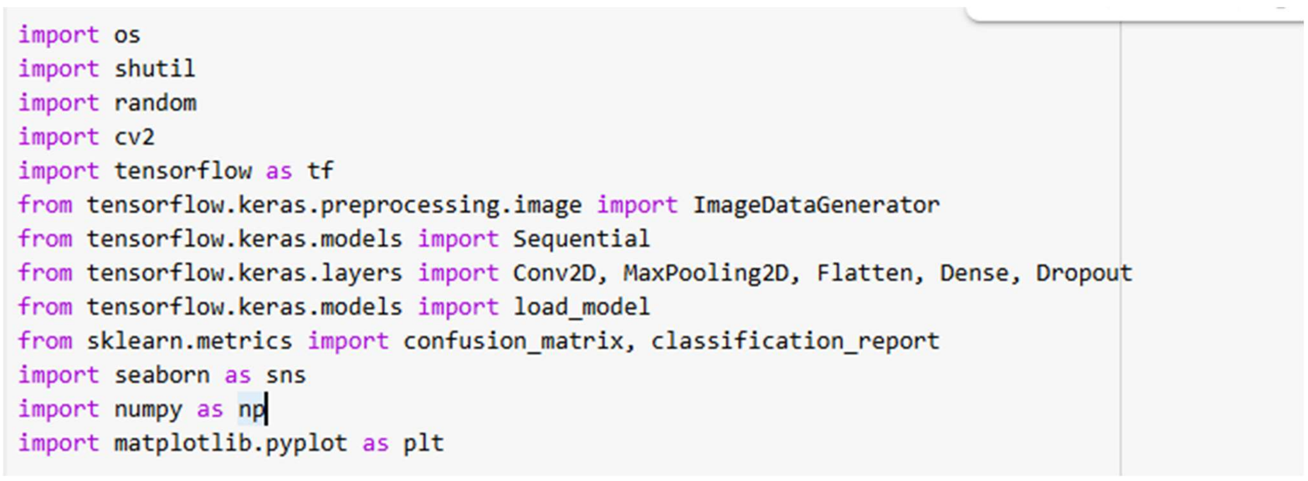
`

CHAPTER 04
IMPLEMENTATION

4.1 PROCESS

A. IMPORTING REQUIRED PACKAGES

Importing packages like os, numpy, cv2, tensorflow etc.



```
import os
import shutil
import random
import cv2
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.models import load_model
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
```

Figure 4.1: Importing Packages

B. DATA ACQUISITION & PROCESSING

1. Dataset Collection

- Source: Publicly available medical imaging repositories such as NIH, Kaggle.
- Classes Covered:

○ Normal (Healthy)	- 820
○ Tuberculosis (TB)	- 487
○ Pneumonia	- 355
○ Lung Cancer (Nodules)	- 417

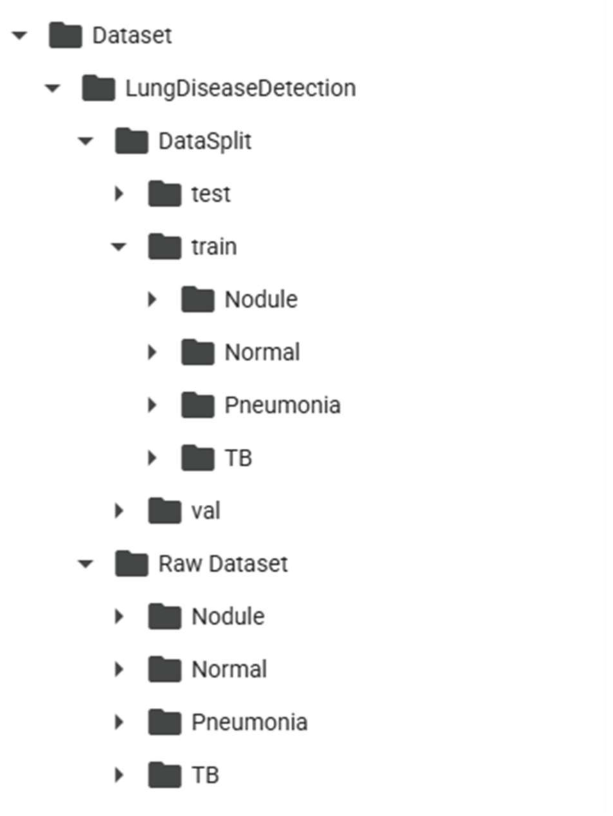
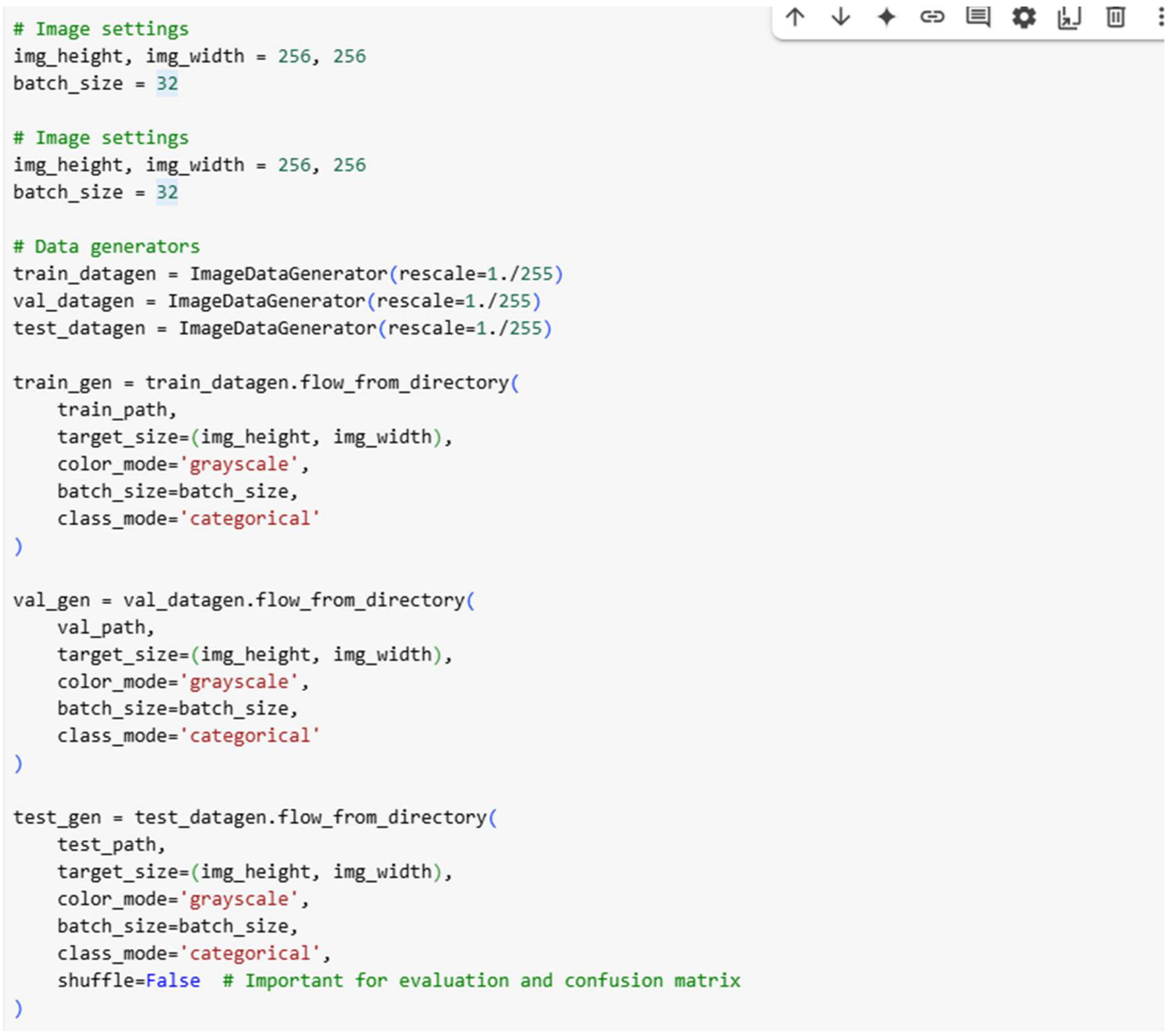


Figure 4.2: Dataset Structure

2. Image Preprocessing

- **Techniques Used:**
    - Conversion to Grayscale for uniformity.
    - Resizing all images to a fixed dimension (e.g., 224×224) for model compatibility.
    - Histogram Equalization & CLAHE (Contrast Limited Adaptive Histogram Equalization) for contrast enhancement.
    - Normalization (pixel scaling between 0 and 1).
- **Augmentation:**
    - Applied using Keras ImageDataGenerator to improve model generalization:
        - Random rotation, flipping, zooming, and shifting.

`



```
# Image settings
img_height, img_width = 256, 256
batch_size = 32

# Image settings
img_height, img_width = 256, 256
batch_size = 32

# Data generators
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_gen = train_datagen.flow_from_directory(
    train_path,
    target_size=(img_height, img_width),
    color_mode='grayscale',
    batch_size=batch_size,
    class_mode='categorical'
)

val_gen = val_datagen.flow_from_directory(
    val_path,
    target_size=(img_height, img_width),
    color_mode='grayscale',
    batch_size=batch_size,
    class_mode='categorical'
)

test_gen = test_datagen.flow_from_directory(
    test_path,
    target_size=(img_height, img_width),
    color_mode='grayscale',
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False  # Important for evaluation and confusion matrix
)
```

Figure 4.3: Image Enhancement

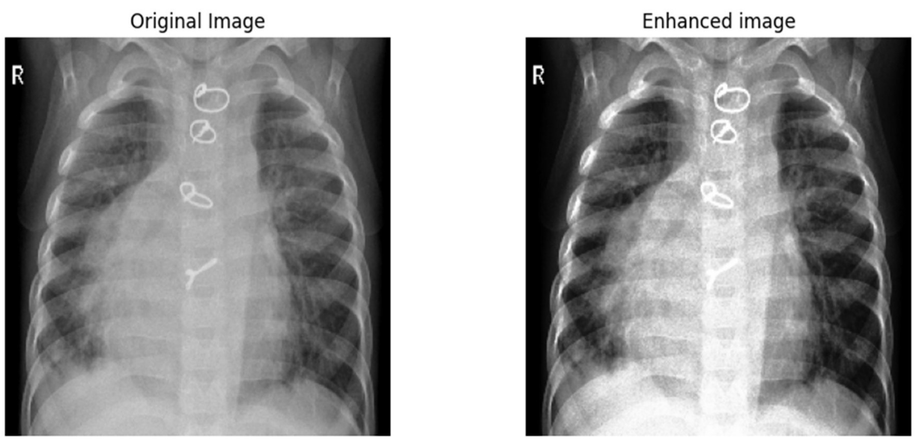
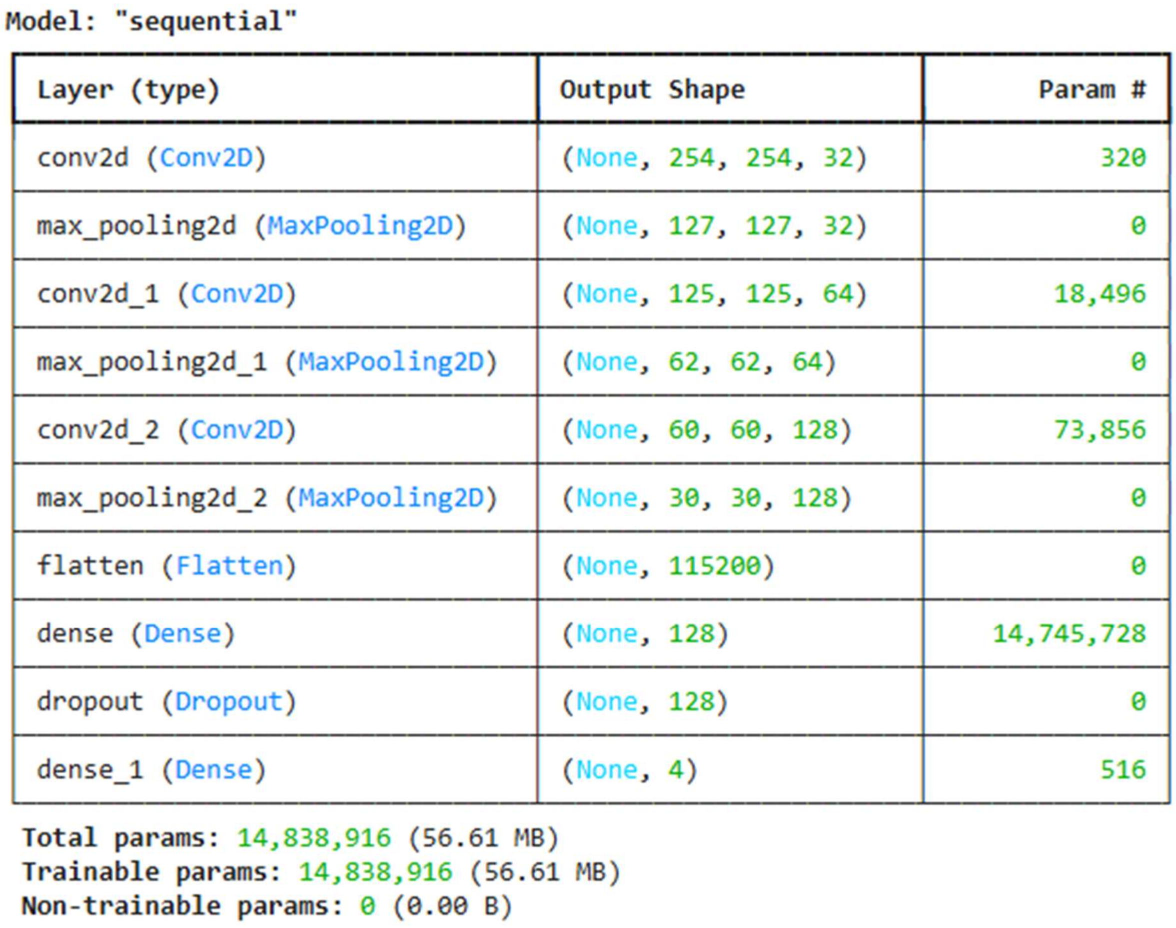


Figure 4.4: Image Enhancement Before vs After

`

C. DEEP LEARNING MODEL ARCHITECTURE



Model: "sequential"

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 254, 254, 32)	320
max_pooling2d (MaxPooling2D)	(None, 127, 127, 32)	0
conv2d_1 (Conv2D)	(None, 125, 125, 64)	18,496
max_pooling2d_1 (MaxPooling2D)	(None, 62, 62, 64)	0
conv2d_2 (Conv2D)	(None, 60, 60, 128)	73,856
max_pooling2d_2 (MaxPooling2D)	(None, 30, 30, 128)	0
flatten (Flatten)	(None, 115200)	0
dense (Dense)	(None, 128)	14,745,728
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 4)	516

```
Total params: 14,838,916 (56.61 MB)
Trainable params: 14,838,916 (56.61 MB)
Non-trainable params: 0 (0.00 B)
```

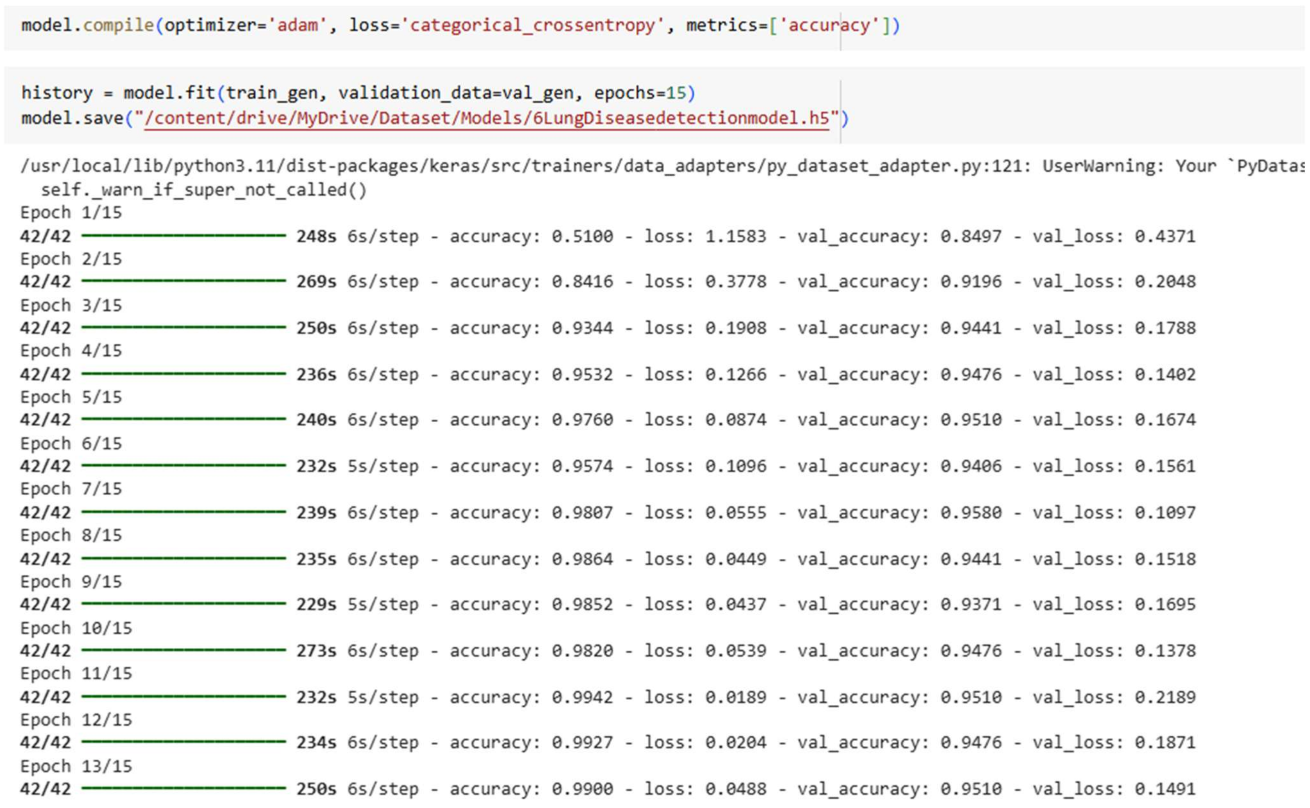
Figure 4.5: CNN Architecture

1. Model Structure

- **Main Classification Model:**
  - A CNN-based architecture trained on all four classes.
- **Sub-models:**
  - **Pneumonia vs TB Classifier:** Improves granularity between these similar classes.
  - **Nodule Detector (Cancer Focused):** Trained specifically to detect lung cancer patterns.

2. Training Details

`



```
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
history = model.fit(train_gen, validation_data=val_gen, epochs=15)
model.save("/content/drive/MyDrive/Dataset/Models/6LungDiseasedetectionmodel.h5")
```

```
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDatas
  self._warn_if_super_not_called()
Epoch 1/15
42/42 ━━━━━━━━━━━━━━━━━━━━ 248s 6s/step - accuracy: 0.5100 - loss: 1.1583 - val_accuracy: 0.8497 - val_loss: 0.4371
Epoch 2/15
42/42 ━━━━━━━━━━━━━━━━━━━━ 269s 6s/step - accuracy: 0.8416 - loss: 0.3778 - val_accuracy: 0.9196 - val_loss: 0.2048
Epoch 3/15
42/42 ━━━━━━━━━━━━━━━━━━━━ 250s 6s/step - accuracy: 0.9344 - loss: 0.1908 - val_accuracy: 0.9441 - val_loss: 0.1788
Epoch 4/15
42/42 ━━━━━━━━━━━━━━━━━━━━ 236s 6s/step - accuracy: 0.9532 - loss: 0.1266 - val_accuracy: 0.9476 - val_loss: 0.1402
Epoch 5/15
42/42 ━━━━━━━━━━━━━━━━━━━━ 240s 6s/step - accuracy: 0.9760 - loss: 0.0874 - val_accuracy: 0.9510 - val_loss: 0.1674
Epoch 6/15
42/42 ━━━━━━━━━━━━━━━━━━━━ 232s 5s/step - accuracy: 0.9574 - loss: 0.1096 - val_accuracy: 0.9406 - val_loss: 0.1561
Epoch 7/15
42/42 ━━━━━━━━━━━━━━━━━━━━ 239s 6s/step - accuracy: 0.9807 - loss: 0.0555 - val_accuracy: 0.9580 - val_loss: 0.1097
Epoch 8/15
42/42 ━━━━━━━━━━━━━━━━━━━━ 235s 6s/step - accuracy: 0.9864 - loss: 0.0449 - val_accuracy: 0.9441 - val_loss: 0.1518
Epoch 9/15
42/42 ━━━━━━━━━━━━━━━━━━━━ 229s 5s/step - accuracy: 0.9852 - loss: 0.0437 - val_accuracy: 0.9371 - val_loss: 0.1695
Epoch 10/15
42/42 ━━━━━━━━━━━━━━━━━━━━ 273s 6s/step - accuracy: 0.9820 - loss: 0.0539 - val_accuracy: 0.9476 - val_loss: 0.1378
Epoch 11/15
42/42 ━━━━━━━━━━━━━━━━━━━━ 232s 5s/step - accuracy: 0.9942 - loss: 0.0189 - val_accuracy: 0.9510 - val_loss: 0.2189
Epoch 12/15
42/42 ━━━━━━━━━━━━━━━━━━━━ 234s 6s/step - accuracy: 0.9927 - loss: 0.0204 - val_accuracy: 0.9476 - val_loss: 0.1871
Epoch 13/15
42/42 ━━━━━━━━━━━━━━━━━━━━ 250s 6s/step - accuracy: 0.9900 - loss: 0.0488 - val_accuracy: 0.9510 - val_loss: 0.1491
```

Figure 4.6: Model Training details

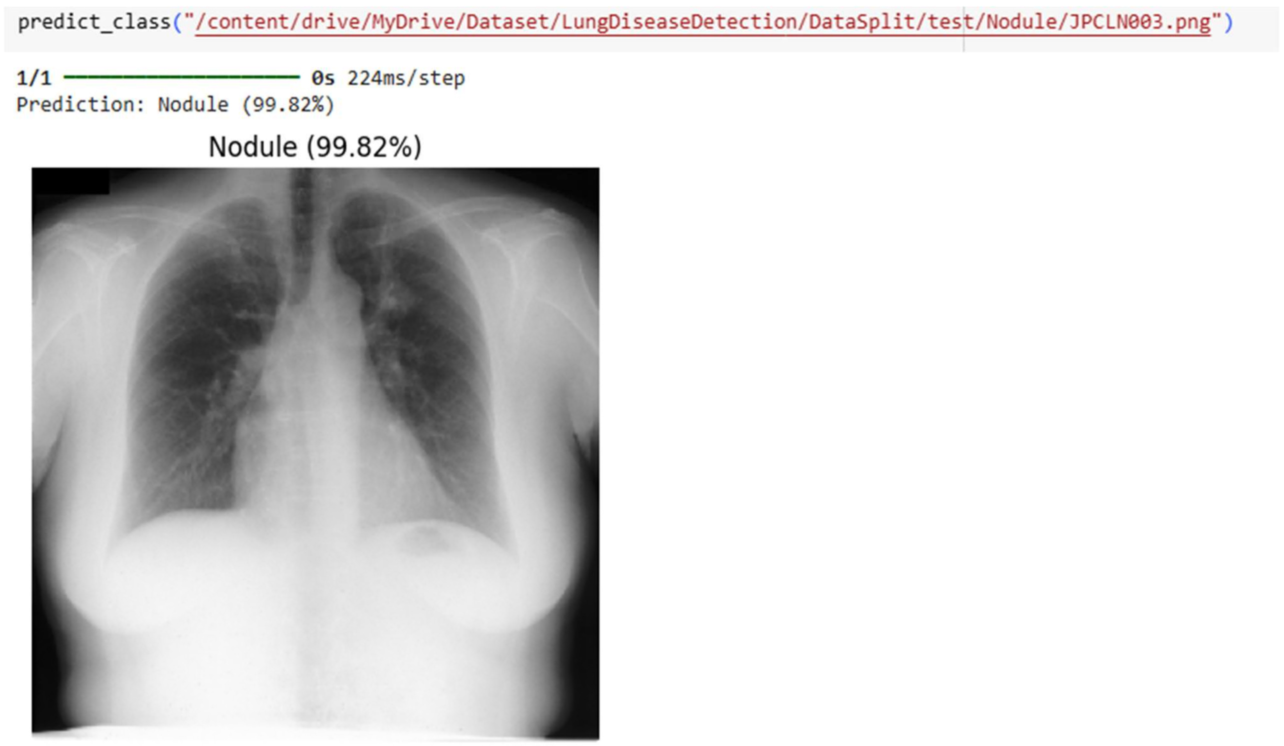
- **Frameworks Used:** TensorFlow / Keras
- **Hyperparameters:**
    - Optimizer: Adam
    - Loss Function: Categorical Crossentropy
    - Batch Size: 32
    - Epochs: 15
- **Validation:** Split into train, validation, and test sets (e.g., 70-20-10).

D. MAKE PREDICTIONS

1. Visual Feedback.:

- **Overlay Technique:** The heatmap is superimposed on the original X-ray for visual interpretability.
- **Prediction Details**: Shows the predicted disease and confidence

`



```
predict_class("/content/drive/MyDrive/Dataset/LungDiseaseDetection/DataSplit/test/Nodule/JPCLN003.png")
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 224ms/step
Prediction: Nodule (99.82%)
```

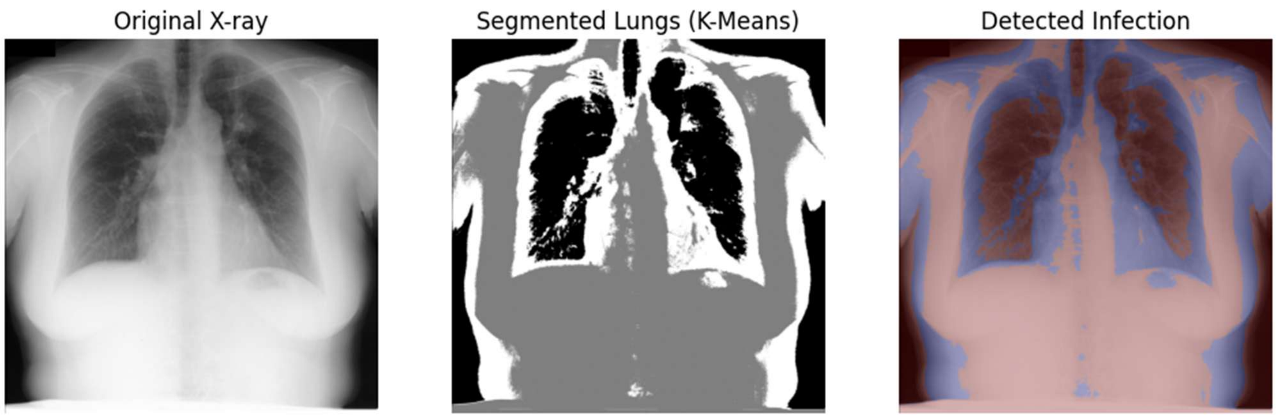


Figure 4.7: Predictions and visualization

E. WEB APPLICATION INTEGRATION

1. Backend Development

- **Framework:** Flask (Python)
- **Functionality:**
    - Handles file upload.
    - Preprocesses image and sends them to the appropriate model.
    - Returns predictions and (if applicable) visualization overlays.

2. Frontend Interface

`

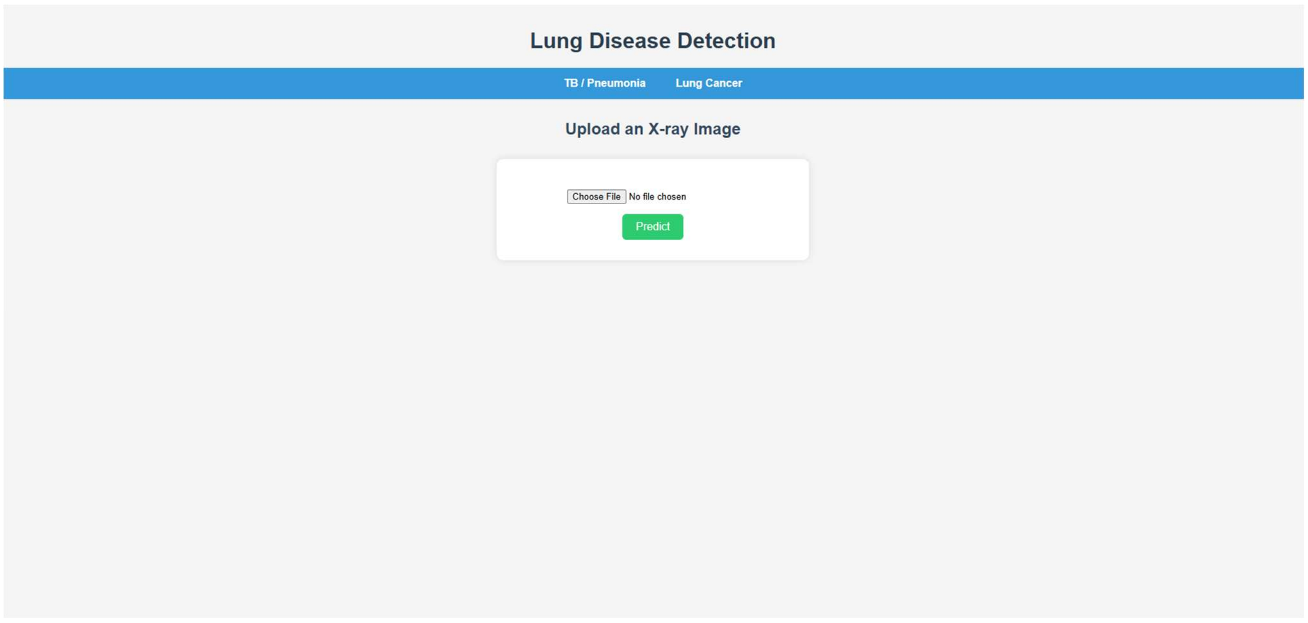


Figure 4.8: Web Interface

- **Technologies Used:** HTML, CSS
- **Features:**
    - Simple UI for X-ray upload.
    - Displays class-wise prediction.
    - Shows infection region (Grad-CAM visualization).

4. Backend-Frontend Integration

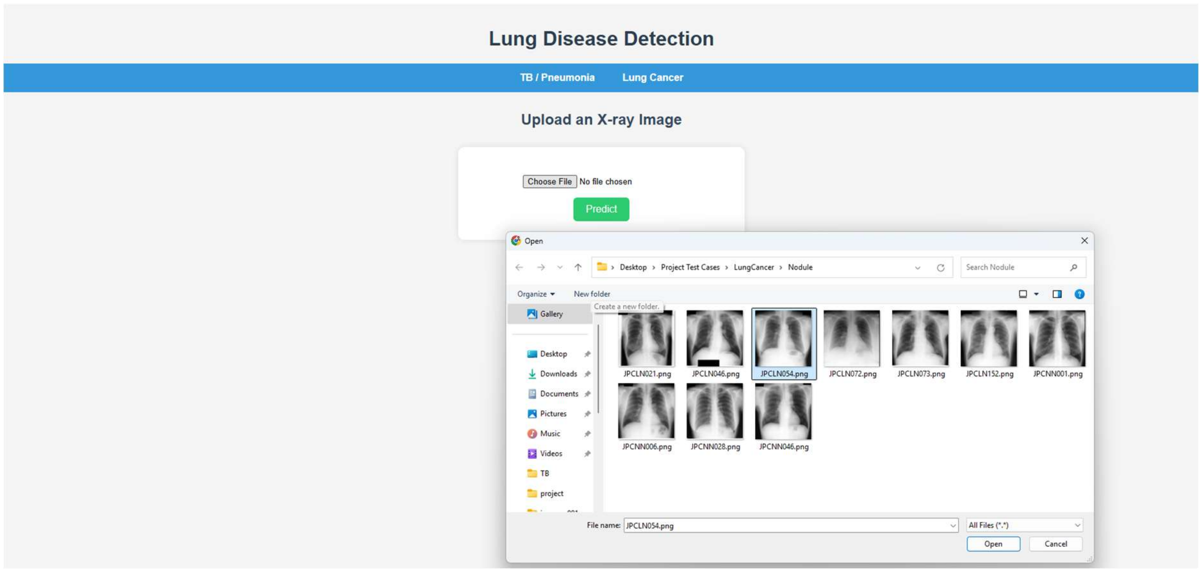
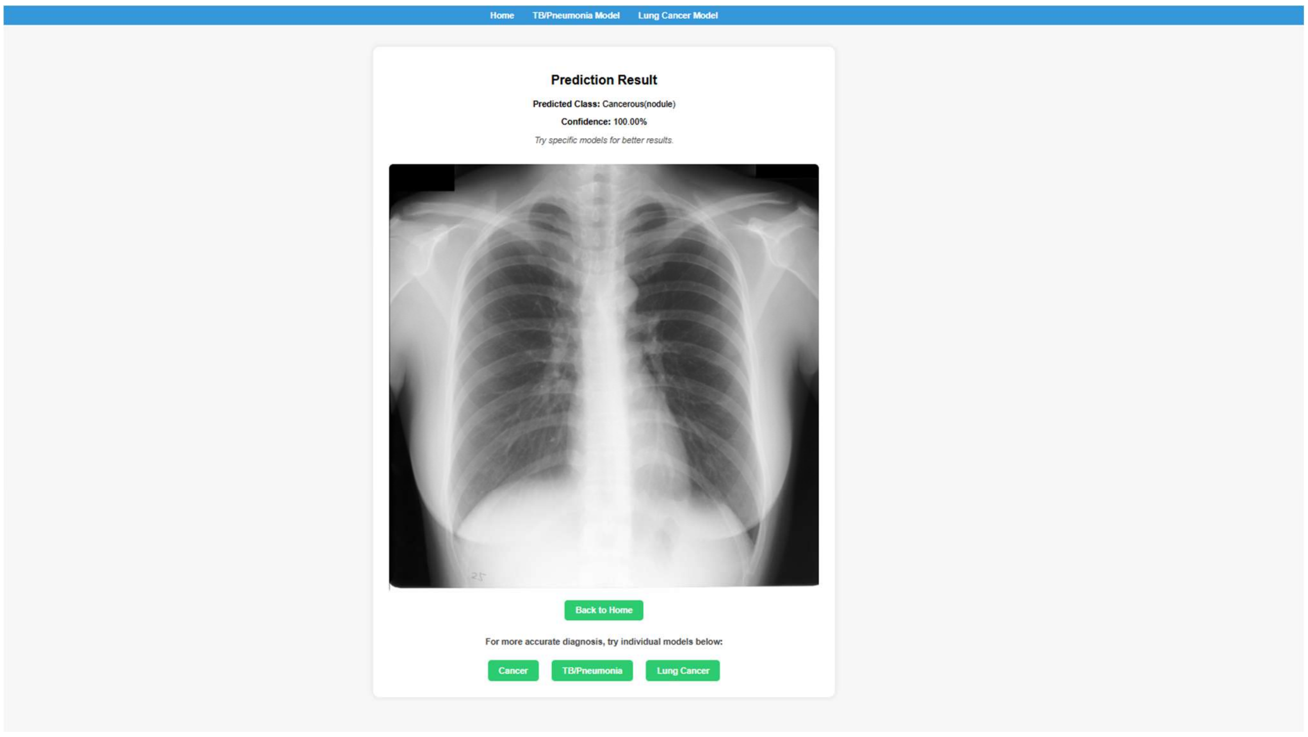


Figure 4.9: Connected backend and frontend

`



Prediction Result

Predicted Class: Cancerous(nodule)

Confidence: 100.00%

Try specific models for better results.

For more accurate diagnosis, try individual models below:

Figure 4.10: Displaying Result

- The frontend and backend are connected using Flask's Jinja templating engine.
- POST requests are used to send user input to the model.
- Prediction results and images are rendered dynamically on the result page.

F. TESTING AND DEPLOYMENT

1.Local Testing

- Unit tests conducted for model accuracy and image upload logic.
- Web interface tested across browsers (Chrome, Firefox).

`

```
# Plot training & validation accuracy
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
```

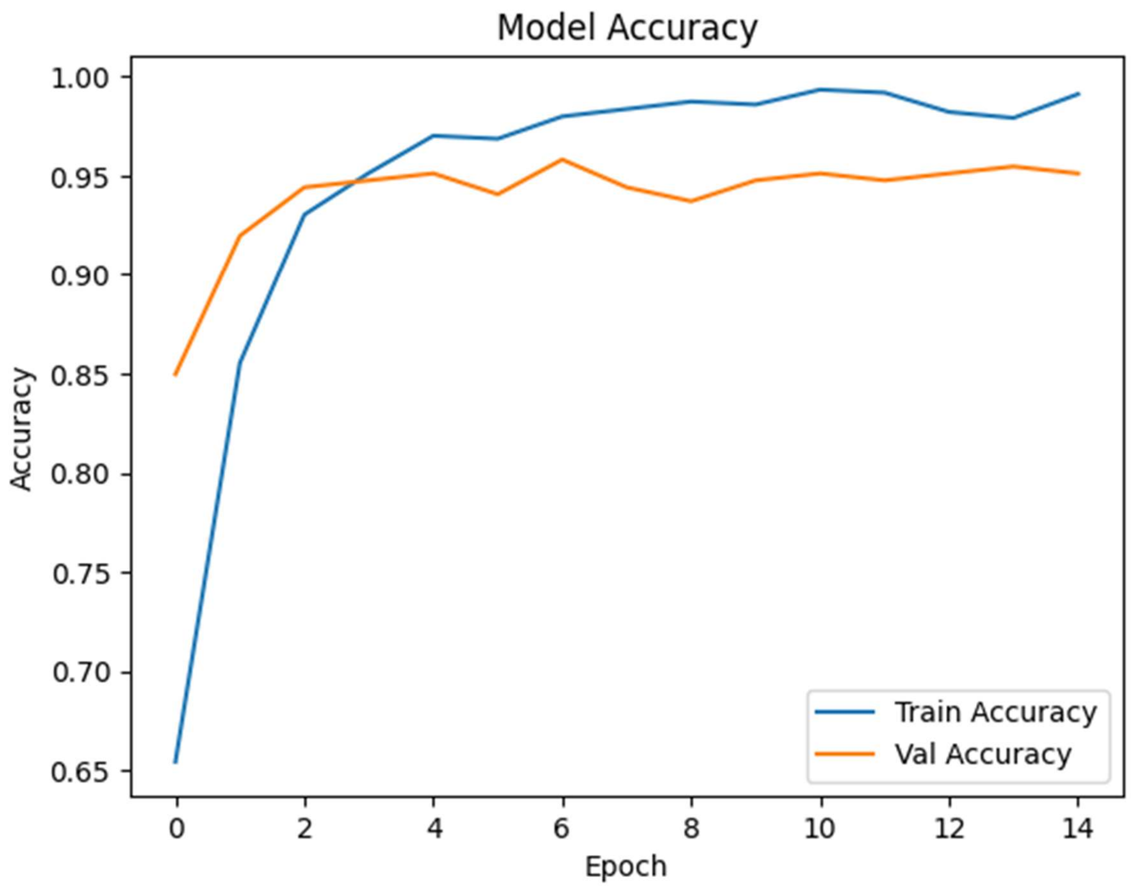
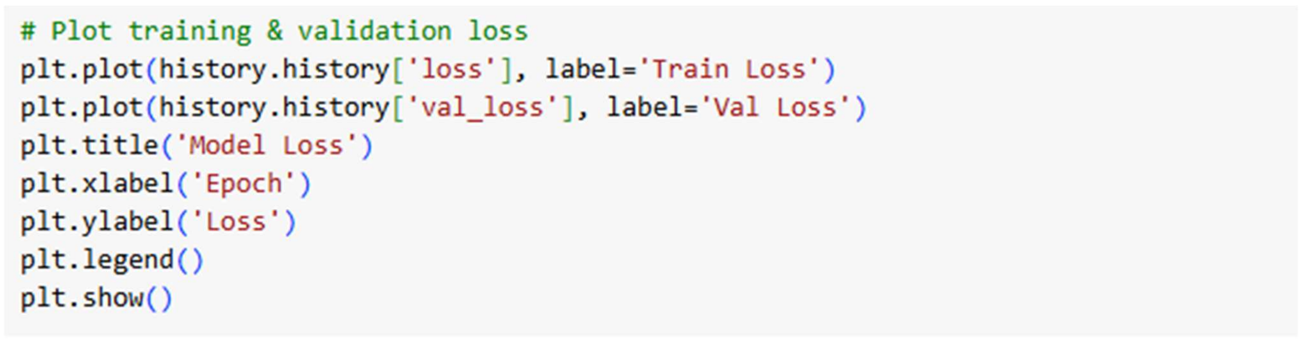


Figure 4.11: Model Training Accuracy and Validation Accuracy

`



```python
# Plot training & validation loss
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
```

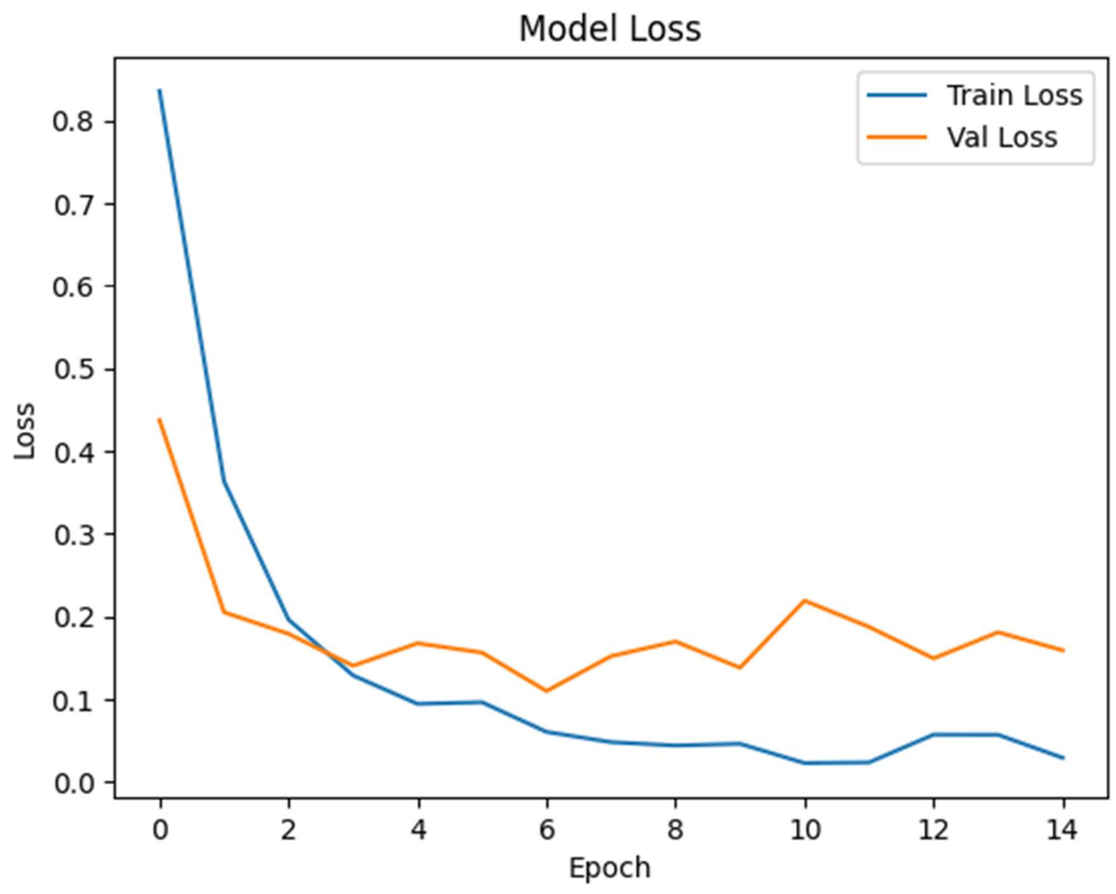
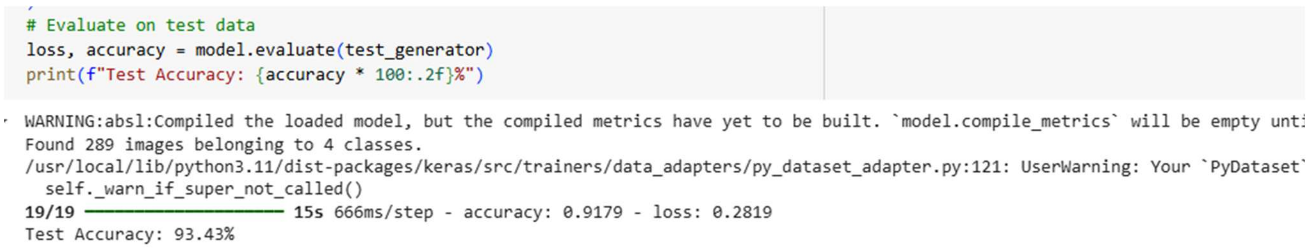


Figure 4.12: Model Loss and Validation loss

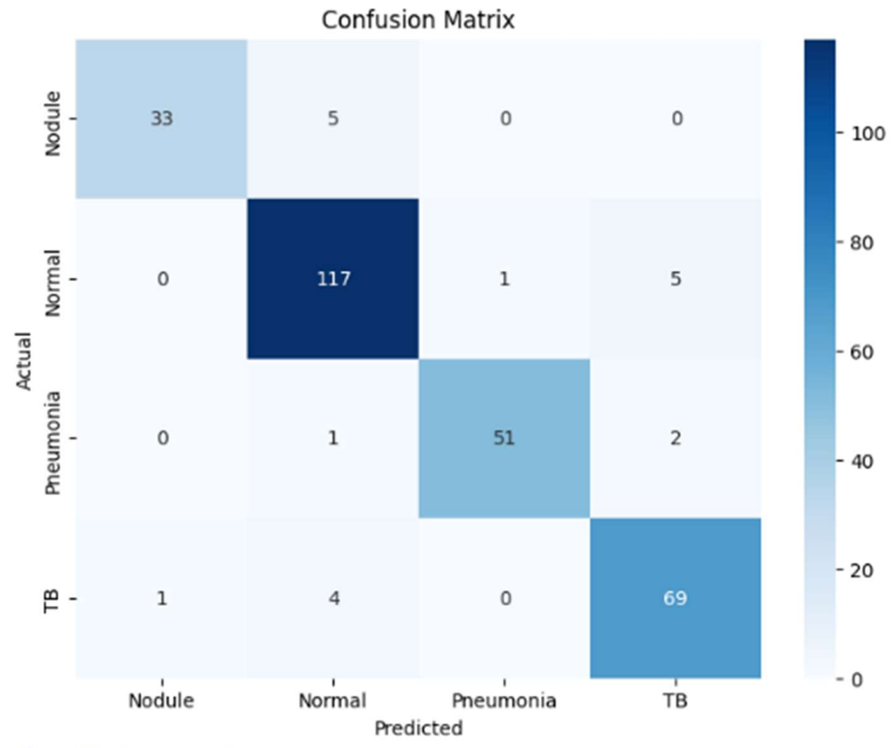


```python
# Evaluate on test data
loss, accuracy = model.evaluate(test_generator)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
WARNING:absl:Compiled the loaded model, but the compiled metrics have yet to be built. `model.compile_metrics` will be empty unti
Found 289 images belonging to 4 classes.
/usr/local/lib/python3.11/dist-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:121: UserWarning: Your `PyDataset`
  self._warn_if_super_not_called()
19/19 ━━━━━━━━━━━━━━━━━━━━ 15s 666ms/step - accuracy: 0.9179 - loss: 0.2819
Test Accuracy: 93.43%
```

Figure 4.13: Model Test Accuracy

`



Actual \ Predicted	Nodule	Normal	Pneumonia	TB
Nodule	33	5	0	0
Normal	0	117	1	5
Pneumonia	0	1	51	2
TB	1	4	0	69

```
Classification Report:

              precision    recall  f1-score   support

      Nodule       0.97      0.87      0.92        38
      Normal       0.92      0.95      0.94       123
   Pneumonia       0.98      0.94      0.96        54
          TB       0.91      0.93      0.92        74

    accuracy                           0.93       289
   macro avg       0.95      0.92      0.93       289
weighted avg       0.94      0.93      0.93       289
```

Figure 4.14: Confusion Matrix for Test Data

2. Deployment (Optional)

- **Platform:** Render / Heroku / Local server
- **Final Features:**
  - Upload → Predict → Visualize → Download report (optional)

`

4.2 SAMPLE CODE:

MODEL DEVELOPMENT:

```
import os
import shutil
import random
import cv2
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout
from tensorflow.keras.models import load_model
from sklearn.metrics import confusion_matrix, classification_report
import seaborn as sns
import numpy as np
import matplotlib.pyplot as plt
# Paths
source_dir = "/content/drive/MyDrive/Dataset/LungDiseaseDetection/Raw Dataset"
target_dir = "/content/drive/MyDrive/Dataset/LungDiseaseDetection/DataSplit"

classes = ["Nodule", "Normal", "Pneumonia", "TB"]
split_ratios = (0.7, 0.15, 0.15)  # train, val, test

for cls in classes:
    cls_path = os.path.join(source_dir, cls)
    images = os.listdir(cls_path)
    random.shuffle(images)

    train_cutoff = int(split_ratios[0] * len(images))
    val_cutoff = train_cutoff + int(split_ratios[1] * len(images))

    split_data = {
        "train": images[:train_cutoff],
        "val": images[train_cutoff:val_cutoff],
        "test": images[val_cutoff:]
    }

    for split in split_data:
        split_dir = os.path.join(target_dir, split, cls)
        os.makedirs(split_dir, exist_ok=True)

        for img_name in split_data[split]:
            src_path = os.path.join(cls_path, img_name)
            dst_path = os.path.join(split_dir, img_name)
            shutil.copy(src_path, dst_path)

print(" Dataset split completed successfully")
# Data Loading
```

`

```
train_path = "/content/drive/MyDrive/Dataset/LungDiseaseDetection/DataSplit/train"
val_path = "/content/drive/MyDrive/Dataset/LungDiseaseDetection/DataSplit/val"
test_path = "/content/drive/MyDrive/Dataset/LungDiseaseDetection/DataSplit/test"
# Image settings
img_height, img_width = 256, 256
batch_size = 32

# Image settings
img_height, img_width = 256, 256
batch_size = 32

# Data generators
train_datagen = ImageDataGenerator(rescale=1./255)
val_datagen = ImageDataGenerator(rescale=1./255)
test_datagen = ImageDataGenerator(rescale=1./255)

train_gen = train_datagen.flow_from_directory(
    train_path,
    target_size=(img_height, img_width),
    color_mode='grayscale',
    batch_size=batch_size,
    class_mode='categorical'
)

val_gen = val_datagen.flow_from_directory(
    val_path,
    target_size=(img_height, img_width),
    color_mode='grayscale',
    batch_size=batch_size,
    class_mode='categorical'
)

test_gen = test_datagen.flow_from_directory(
    test_path,
    target_size=(img_height, img_width),
    color_mode='grayscale',
    batch_size=batch_size,
    class_mode='categorical',
    shuffle=False  # Important for evaluation and confusion matrix
)
# Visualize one sample (original vs CLAHE-enhanced)
img = next(train_gen)[0][0].squeeze()  # Get first image from batch
original = img.copy()

img_uint8 = (original * 255).astype(np.uint8)
enhanced = cv2.equalizeHist(img_uint8)

plt.figure(figsize=(10, 4))
plt.subplot(1, 2, 1)
plt.imshow(original, cmap='gray')
```

`

```
plt.title("Original Image")
plt.axis('off')

plt.subplot(1, 2, 2)
plt.imshow(enhanced, cmap='gray')
plt.title("Enhanced image")
plt.axis('off')
plt.tight_layout()
plt.show()
# Build CNN
model = Sequential([
    Conv2D(32, (3,3), activation='relu', input_shape=(256, 256, 1)),
    MaxPooling2D(2,2),

    Conv2D(64, (3,3), activation='relu'),
    MaxPooling2D(2,2),

    Conv2D(128, (3,3), activation='relu'),
    MaxPooling2D(2,2),

    Flatten(),
    Dense(128, activation='relu'),
    Dropout(0.3),
    Dense(4, activation='softmax')  # 4 classes
])
model.summary()
model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
history = model.fit(train_gen, validation_data=val_gen, epochs=15)
model.save("/content/drive/MyDrive/Dataset/Models/6LungDiseasedetectionmodel.h5")
# Plot training & validation accuracy
plt.plot(history.history['accuracy'], label='Train Accuracy')
plt.plot(history.history['val_accuracy'], label='Val Accuracy')
plt.title('Model Accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend()
plt.show()
# Plot training & validation loss
plt.plot(history.history['loss'], label='Train Loss')
plt.plot(history.history['val_loss'], label='Val Loss')
plt.title('Model Loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.legend()
plt.show()
# Path to your test data
test_dir = '/content/drive/MyDrive/Dataset/LungDiseaseDetection/DataSplit/test'

# Load the saved model (if you haven’t already)
model = load_model('/content/drive/MyDrive/Dataset/Models/6LungDiseasedetectionmodel.h5')
```

`

```
# Define test data generator (no augmentation, just rescale)
test_datagen = ImageDataGenerator(rescale=1./255)

# Create the test generator
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=(256, 256),  # Make sure this matches training size
    batch_size=16,
    class_mode='categorical',  # because we have 5 classes
    shuffle=False,
    color_mode='grayscale'  # This is the line to be added
)
# Evaluate on test data
loss, accuracy = model.evaluate(test_generator)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
# Get predictions
predictions = model.predict(test_generator)
predicted_classes = np.argmax(predictions, axis=1)
true_classes = test_generator.classes
class_labels = list(test_generator.class_indices.keys())

# Confusion Matrix
cm = confusion_matrix(true_classes, predicted_classes)

plt.figure(figsize=(8,6))
sns.heatmap(cm,        annot=True,        fmt="d",        cmap="Blues",        xticklabels=class_labels,
yticklabels=class_labels)
plt.xlabel("Predicted")
plt.ylabel("Actual")
plt.title("Confusion Matrix")
plt.show()

# Classification Report
report = classification_report(true_classes, predicted_classes, target_names=class_labels)
print(" Classification Report:\n")
print(report)
# Load the classification model

classification_model                                                                    =
load_model('/content/drive/MyDrive/Dataset/Models/6LungDiseasedetectionmodel.h5')

# Class names (update if your class order is different)
class_names =  ["Nodule", "Normal", "Pneumonia", "TB"]

def predict_class(image_path):
    # Load and preprocess image
    img = cv2.imread(image_path, cv2.IMREAD_GRAYSCALE)
    img_resized = cv2.resize(img, (256, 256))
    img_normalized = img_resized / 255.0
    input_img = np.expand_dims(img_normalized, axis=-1)
    input_img = np.expand_dims(input_img, axis=0)
```

`

```
    # Predict
    prediction = classification_model.predict(input_img)[0]
    predicted_class = np.argmax(prediction)
    confidence = prediction[predicted_class] * 100

    # Print result
    print(f"Prediction: {class_names[predicted_class]} ({confidence:.2f}%)")

    # Show the image with prediction title
    plt.imshow(img_resized, cmap='gray')
    plt.title(f"{class_names[predicted_class]} ({confidence:.2f}%)")
    plt.axis('off')
    plt.show()
```

WEB INTERFACE CODE:

App.py (Flask):

```
from PIL.ImageOps import grayscale
from flask import Flask, render_template, request, url_for
import os
import cv2
import numpy as np
from tensorflow.keras.models import load_model
from werkzeug.utils import secure_filename
from infected import visualize_infection, display_predicted_mask  # your infection overlay
function

app = Flask(__name__)

# Configure upload folder
UPLOAD_FOLDER = os.path.join('static', 'uploads')
os.makedirs(UPLOAD_FOLDER, exist_ok=True)
app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

# Load models
tb_pneumonia_model = load_model('models/TBPneumoniaClassificationCNN.h5')
lung_cancer_model = load_model('models/NoduleDetectionModel.h5')
main_model = load_model('models/6LungDiseasedetectionmodel.h5')

# Preprocess function
def preprocess_image(img_path):
    img = cv2.imread(img_path, cv2.IMREAD_GRAYSCALE)  # Read grayscale
    img = cv2.resize(img, (256, 256))  # Resize to match model input
    img = img / 255.0  # Normalize
    img = np.expand_dims(img, axis=-1)
    img = np.expand_dims(img, axis=0)
    return img.reshape(1, 256, 256, 1)  # Shape: (1, 256, 256, 1)

# Homepage
@app.route('/')
def index():
```

`

```
    return render_template('index.html')

# Main model prediction
@app.route('/predict', methods=['POST'])
def predict():
    file = request.files['image']
    filename = secure_filename(file.filename)
    upload_folder = os.path.join('static', 'uploads')
    os.makedirs(upload_folder, exist_ok=True)

    filepath = os.path.join(upload_folder, filename)
    file.save(filepath)

    img = preprocess_image(filepath)
    pred = main_model.predict(img)[0]
    classes = ['Cancerous(nodule)', 'Normal','Pneumonia', 'TB']
    result = classes[np.argmax(pred)]
    confidence = pred[np.argmax(pred)] * 100

    return render_template("result.html",
                           prediction=result,
                           confidence=f"{confidence:.2f}%",
                           image_path=url_for('static', filename=f"uploads/{filename}"),
                           extra_msg="Try specific models for better results.")



# TB / Pneumonia model route
@app.route('/tb_pneumonia', methods=['GET', 'POST'])
def tb_pneumonia():
    if request.method == 'POST':
        file = request.files['image']
        filename = secure_filename(file.filename)
        filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(filepath)

        img = preprocess_image(filepath)
        pred = tb_pneumonia_model.predict(img)[0]
        classes = ['Normal', 'Pneumonia', 'TB']
        result = classes[np.argmax(pred)]
        confidence = pred[np.argmax(pred)] * 100

        return render_template("result.html",
                               prediction=result,
                               confidence=f"{confidence:.2f}%",
                               image_path=url_for('static', filename=f"uploads/{filename}"))

    return render_template("tb_pneumonia.html")

# Lung cancer model route
@app.route('/lung_cancer', methods=['GET', 'POST'])
def lung_cancer():
```

`

```
    if request.method == 'POST':
        file = request.files['image']
        filename = secure_filename(file.filename)
        filepath = os.path.join(app.config['UPLOAD_FOLDER'], filename)
        file.save(filepath)

        img = preprocess_image(filepath)
        pred = lung_cancer_model.predict(img)[0]
        classes = ['Nodule(cancerous)' ,'Normal']
        result = classes[np.argmax(pred)]
        confidence = pred[np.argmax(pred)] * 100

        return render_template("result.html",
                               prediction=result,
                               confidence=f"{confidence:.2f}%",
                               image_path=url_for('static', filename=f"uploads/{filename}"))

    return render_template("lung_cancer.html")

if __name__ == '__main__':
    app.run(debug=True)
```

Index.html

```
<!DOCTYPE html>
<html>
<head>
    <title>Lung Disease Detection</title>
    <link rel="stylesheet" type="text/css" href="{{ url_for('static', filename='css/style.css') }}">
</head>
<body>
    <h1>Lung Disease Detection</h1>

    <ul>
        <li><a href="/tb_pneumonia">TB / Pneumonia</a></li>
        <li><a href="/lung_cancer">Lung Cancer</a></li>
    </ul>

    <h2>Upload an X-ray Image</h2>
    <form method="POST" action="/predict" enctype="multipart/form-data">
        <input type="file" name="image" required><br>
        <input type="submit" value="Predict">
    </form>
</body>
</html>
```

Lung_cancer.html

```
<!DOCTYPE html>
<html>
```

`

```
<head>
   <title>Lung Cancer</title>
   <link rel="stylesheet" href="{{ url_for('static', filename='css/lung_cancer.css') }}">
</head>
<body>
<nav>
   <li><a href="/">Home</a></li>
   <li><a href="/tb_pneumonia">TB / Pneumonia</a></li>
   <li><a href="/lung_cancer">Lung Cancer</a></li>
</nav>
<hr>
   <h1>Lung Cancer Classification</h1>
   <form action="/lung_cancer" method="post" enctype="multipart/form-data">
      <input type="file" name="image" required><br><br>
      <button type="submit">Predict Lung Cancer</button>
   </form>
</body>
</html>
```

Tb_pneumonia.html

```
<!DOCTYPE html>
<html>
<head>
   <title>TB & Pneumonia</title>
   <link rel="stylesheet" href="{{ url_for('static', filename='css/tb_pneumonia.css') }}">
</head>
<body>
<ul>
   <li><a href="/">Home</a></li>
   <li><a href="/tb_pneumonia">TB / Pneumonia</a></li>
   <li><a href="/lung_cancer">Lung Cancer</a></li>
</ul>
<hr>
   <h1>TB, Normal, Pneumonia Classification</h1>
   <form action="/tb_pneumonia" method="post" enctype="multipart/form-data">
      <input type="file" name="image" required><br><br>
      <button type="submit">Predict TB/Pneumonia</button>
   </form>
</body>
</html>
```

Result/html

```
<!DOCTYPE html>
<html lang="en">
<head>
   <meta charset="UTF-8">
   <title>Prediction Result</title>
   <link rel="stylesheet" href="{{ url_for('static', filename='css/result.css') }}">
</head>
```

`

```
<body>

<!-- Navigation Bar -->
<nav class="navbar">
   <ul>
      <li><a href="/">Home</a></li>
      <li><a href="/tb_pneumonia">TB/Pneumonia Model</a></li>
      <li><a href="/lung_cancer">Lung Cancer Model</a></li>
   </ul>
</nav>

<div class="result-container">
   <h2>Prediction Result</h2>
   <p><strong>Predicted Class:</strong> {{ prediction }}</p>
   <p><strong>Confidence:</strong> {{ confidence }}</p>

   {% if extra_msg %}
      <p class="note">{{ extra_msg }}</p>
   {% endif %}

   {% if image_path %}
      <img src="{{ image_path }}" alt="Uploaded Image">
   {% else %}
      <p>No image to display.</p>
   {% endif %}

   <br>
   <a href="/" class="btn">Back to Home</a>

   <p class="final-suggestion">
      For more accurate diagnosis, try individual models below:
   </p>

   <div class="model-links">
      <a class="model-btn" href="/tb_pneumonia">TB/Pneumonia</a>
      <a class="model-btn" href="/lung_cancer">Lung Cancer</a>
   </div>
</div>

</body>
</html>
```

STYLE.CSS

```
/* Basic Reset */
body {
   font-family: Arial, sans-serif;
   background-color: #f5f5f5;
   margin: 0;
   padding: 0;
}
```

`

```
/* Page Title */
h1 {
   text-align: center;
   color: #2c3e50;
   margin-top: 40px;
}

/* Navigation Bar */
ul {
   list-style-type: none;
   margin: 0;
   padding: 0;
   background-color: #3498db; /* Blue navbar */
   overflow: hidden;
   text-align: center;
}

ul li {
   display: inline;
}

ul li a {
   display: inline-block;
   padding: 14px 20px;
   text-decoration: none;
   color: white;
   font-weight: bold;
}

ul li a:hover {
   background-color: #217dbb;
}

/* Section Title */
h2 {
   text-align: center;
   color: #34495e;
   margin-top: 30px;
}

/* Form Styling */
form {
   background-color: white;
   padding: 30px;
   max-width: 400px;
   margin: 30px auto;
   text-align: center;
   border-radius: 10px;
   box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}
```

`

```
/* File Input */
input[type="file"] {
   margin: 15px 0;
}

/* Submit Button */
input[type="submit"] {
   background-color: #2ecc71;
   color: white;
   padding: 10px 20px;
   font-size: 16px;
   border: none;
   border-radius: 6px;
   cursor: pointer;
}

input[type="submit"]:hover {
   background-color: #27ae60;
}
```

Lung_cancer.css

```
body {
   font-family: 'Segoe UI', sans-serif;
   background-color: #f0f0f0;
   padding: 20px;
}


nav {
   background-color: #34495e;
   padding: 10px;
   text-align: center;
}


nav li {
   display: inline-block;
   margin: 0 15px;
}


nav li a {
   color: #fff;
   text-decoration: none;
   font-weight: 600;
```

`
```
}

h1 {
    text-align: center;
    margin-top: 30px;
    color: #2c3e50;
}

form {
    max-width: 400px;
    margin: 30px auto;
    padding: 25px;
    background-color: #ffffff;
    border-radius: 10px;
    box-shadow: 0 0 10px rgba(0,0,0,0.1);
    text-align: center;
}

input[type="file"] {
    width: 100%;
    margin-bottom: 15px;
    padding: 10px;
}

button {
    padding: 10px 20px;
    background-color: #e67e22;
    color: white;
    font-weight: bold;
    border: none;
    border-radius: 5px;
    cursor: pointer;
}

button:hover {
```

`

```
    background-color: #d35400;
}
```

Tb_pneumonia.css

```
/* Body */
body {
    font-family: Arial, sans-serif;
    background-color: #f5f5f5;
    margin: 0;
    padding: 0;
}

/* Blue Navbar */
ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
    overflow: hidden;
    background-color: #3498db; /* blue */
    text-align: center;
}

ul li {
    display: inline;
}

ul li a {
    display: inline-block;
    color: white;
    padding: 14px 20px;
    text-decoration: none;
    font-weight: bold;
}

ul li a:hover {
```

`

```
    background-color: #217dbb;
}

/* Heading */
h1 {
    text-align: center;
    margin-top: 40px;
    color: #333;
}

/* Form Box */
form {
    background-color: white;
    padding: 30px;
    max-width: 400px;
    margin: 30px auto;
    text-align: center;
    border-radius: 10px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
}

/* File input */
input[type="file"] {
    margin: 10px 0;
}

/* Submit Button */
button {
    background-color: #2ecc71;
    color: white;
    padding: 10px 20px;
    font-size: 16px;
    border: none;
    border-radius: 6px;
    cursor: pointer;
```

`

```
}

button:hover {
    background-color: #27ae60;
}
```

Result.css

```
/* Reset margins and paddings for consistency */
body {
    margin: 0;
    padding: 0;
    font-family: Arial, sans-serif;
    background-color: #f7f7f7;
}

/* Navbar styling */
.navbar {
    background-color: #3498db;
    padding: 10px 0;
    text-align: center;
}

.navbar ul {
    list-style-type: none;
    margin: 0;
    padding: 0;
}

.navbar ul li {
    display: inline;
    margin: 0 15px;
}

.navbar ul li a {
    text-decoration: none;
```

`

```
    color: white;
    font-weight: bold;
}

/* Centering and styling result content */
.result-container {
    max-width: 800px;
    margin: 40px auto;
    padding: 30px;
    background-color: #fff;
    border-radius: 12px;
    box-shadow: 0 0 10px rgba(0, 0, 0, 0.1);
    text-align: center;
}

/* Prediction image */
.result-container img {
    max-width: 100%;
    height: auto;
    border-radius: 8px;
    margin-top: 20px;
}

/* Buttons and links */
.btn,
.model-btn {
    display: inline-block;
    margin: 10px 10px 0;
    padding: 10px 20px;
    background-color: #2ecc71;
    color: white;
    text-decoration: none;
    border-radius: 6px;
    font-weight: bold;
    transition: background-color 0.3s;
```

```
`
 }

 .btn:hover,
 .model-btn:hover {
    background-color: #27ae60;
 }

 /* Note and final suggestion */
 .note {
    font-style: italic;
    color: #555;
    margin-top: 10px;
 }

 .final-suggestion {
    margin-top: 30px;
    font-weight: bold;
    color: #444;
 }
```

4.3 CODE EXECTION

Just click the run symbol in Pycharm application by selecting app.py.
In terminal click on the given IP Address.

TERMINAL PROMPT

After installing all the required packages and libraries, we perform the run operation for for the code to be run in the Pycharm application with the below

- PS C:\Users\bhara> python"C:\Users\ravit\OneDrive\PyCharmProjects\LungDiseasedetection\app.py"

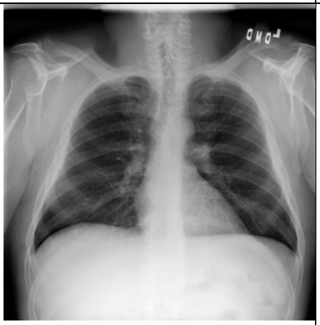
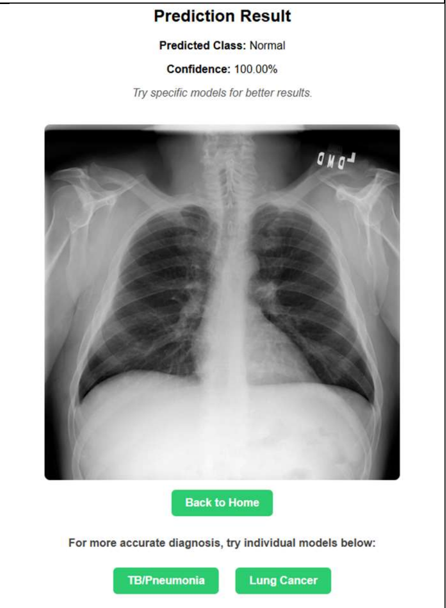
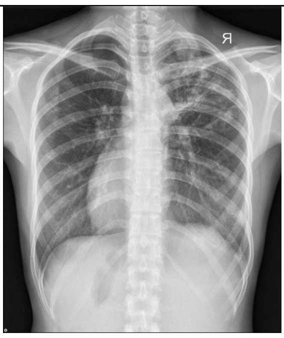
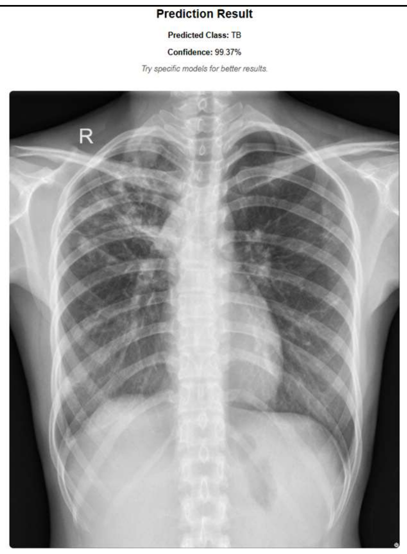
`

CHAPTER 05

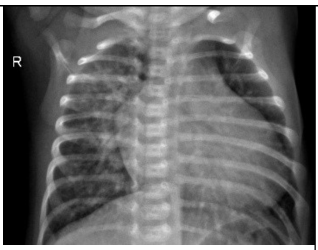
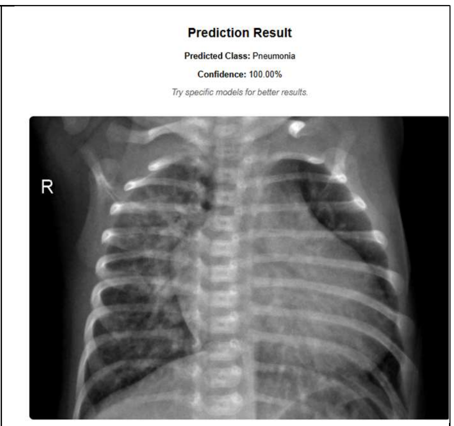
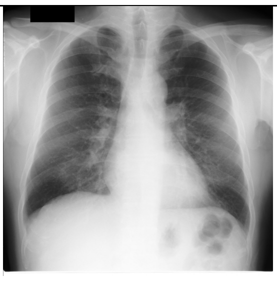
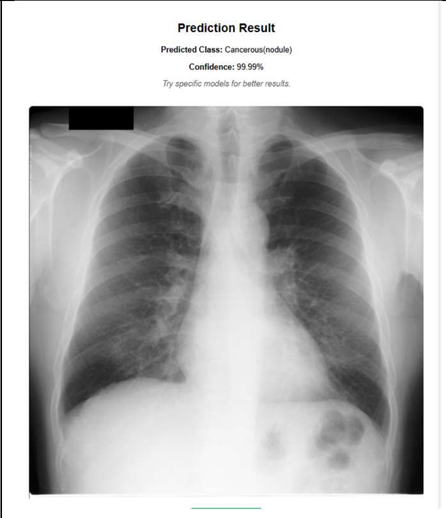
TESTING

5.1 LOCAL TESTING

TABLE 5.1: TEST CASE AND OUTPUT

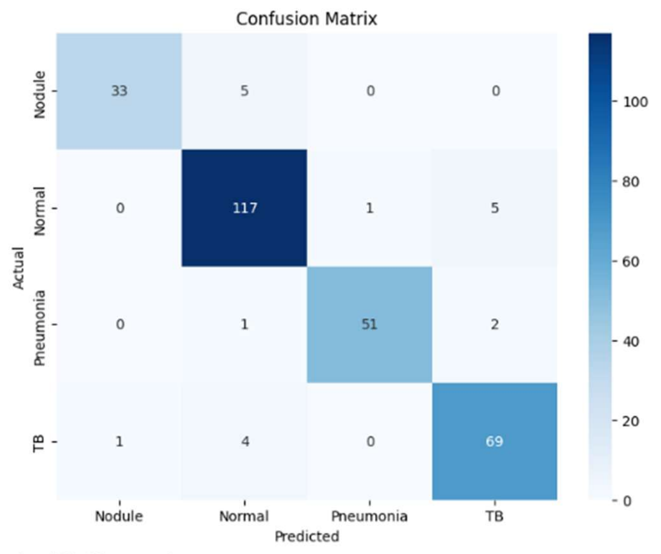
TEST CASE	IMAGE	DISESASE TYPE	PREDICTION	RESULT
Test Case 1		Normal	Normal Confidence: 100%	 **Prediction Result** **Predicted Class:** Normal **Confidence:** 100.00% *Try specific models for better results.* Back to Home For more accurate diagnosis, try individual models below: TB/Pneumonia Lung Cancer
Test Case 2		TB	TB Confidence: 99.37%	 **Prediction Result** **Predicted Class:** TB **Confidence:** 99.37% *Try specific models for better results.*

`

Test Case 3		Pneumonia	Pneumonia Confidence: 100%	 **Prediction Result** Predicted Class: Pneumonia Confidence: 100.00% *Try specific models for better results.*
Test Case 4		Nodule (may cause cancer)	Nodule Confidence: 99.99%	 **Prediction Result** Predicted Class: Cancerous(nodule) Confidence: 99.99% *Try specific models for better results.*

`

TESTING ON A TEST FOLDER :



```
Classification Report:

              precision    recall  f1-score   support

      Nodule       0.97      0.87      0.92        38
      Normal       0.92      0.95      0.94       123
   Pneumonia       0.98      0.94      0.96        54
          TB       0.91      0.93      0.92        74

    accuracy                           0.93       289
   macro avg       0.95      0.92      0.93       289
weighted avg       0.94      0.93      0.93       289
```

Figure 5.1: TESTING ON A TEST FOLDER

- Nodule Test Cases passed 38. In them 33 are predicted as Nodule and remaining 5 are predicted wrong.
- Normal Test Cases passed 123. In them 117 are predicted as Normal and remaining 5 are predicted wrong.
- Pneumonia Test Cases passed 52. In them 51 are predicted as Pneumonia and remaining 1 is predicted wrong.
- TB Test Cases passed 74. In them 69 are predicted as Nodule and remaining 5 are predicted wrong.

`

VISUALIZING INFECTED PART IN GOOGLE COLLAB:

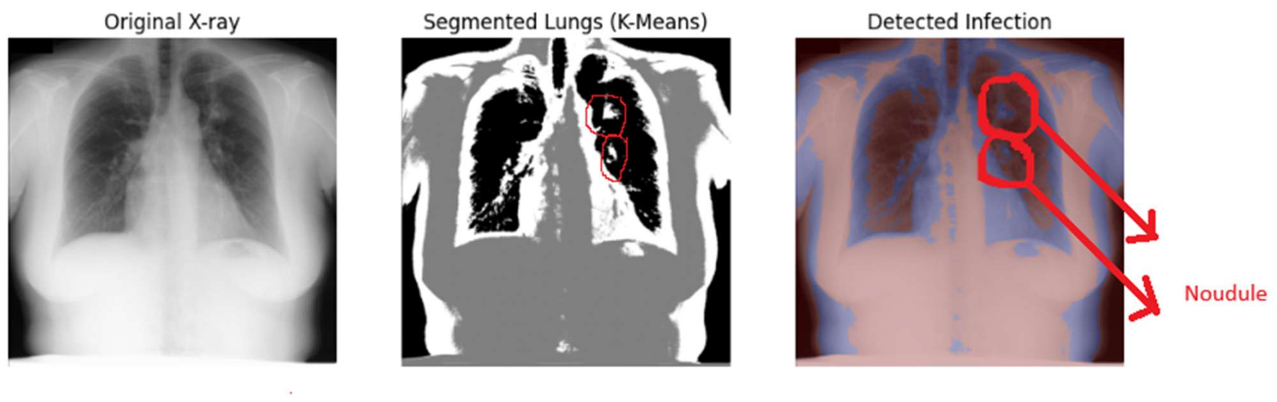


Figure 5.2: Visualization

`

CHAPTER 06

RESULTS AND DISCUSSION

6.1 Performance Metrics

Lung Disease Classification – CNN Model

The deep learning model developed for lung disease detection classifies chest X-ray images into four distinct classes: Nodule, Normal, Pneumonia, and TB. The dataset was split into training, validation, and test sets, and the model's performance was assessed using standard classification metrics.

- Training Accuracy: 98.91%
- Validation Accuracy: 95.45%
- Test Accuracy: 93.43%
- Training Loss: 0.263
- Validation Loss: 0.1807

The classification report on the test set (289 images) is as follows:

Class	Precision	Recall	F1-score	Support
Nodule	0.97	0.87	0.92	38
Normal	0.92	0.95	0.94	123
Pneumonia	0.98	0.94	0.96	54
TB	0.91	0.93	0.92	74

- Overall Accuracy: 93.43%
- Macro Average F1-Score: 0.93
- Weighted Average F1-Score: 0.93

These results indicate a strong and consistent performance across all disease categories, with particularly high precision in Pneumonia and Nodule cases. The model's generalization capability is evident from the balanced recall and F1-scores on the unseen test set.

6.2 Results and Analysis

The lung disease detection system demonstrates high effectiveness in multi-class classification using X-ray imagery. The CNN architecture efficiently extracts visual features critical to disease identification, aided by preprocessing techniques such as image resizing and contrast enhancement.

- Nodule Detection: Despite the smallest support count, the model achieved a high precision (0.97), although with slightly lower recall (0.87), suggesting the need for more diverse samples.

`

- Normal and TB Detection: The model balanced both recall and precision well, confirming robustness in distinguishing healthy from diseased lungs.
- Pneumonia Detection: Achieved one of the highest performance scores, with F1-score of 0.96, indicating that the model reliably detects pneumonia features.

The low validation and test loss values further confirm that the model is not overfitting and is capable of generalizing to unseen images.

UI

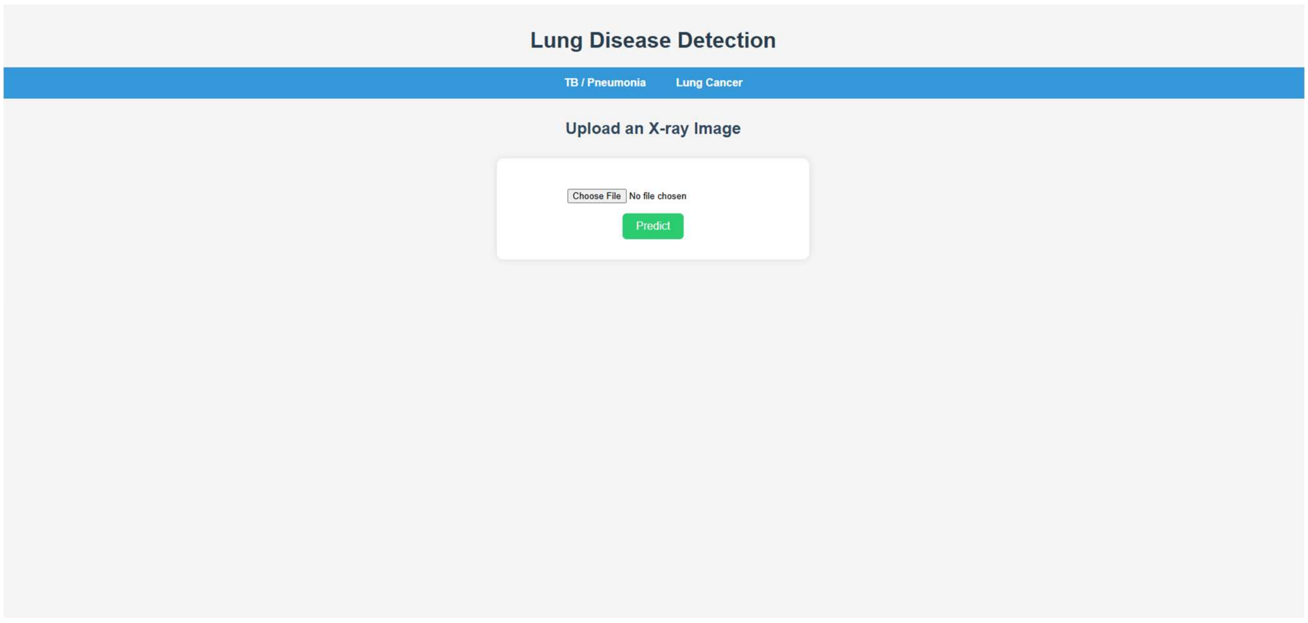


Figure 6.1: UI

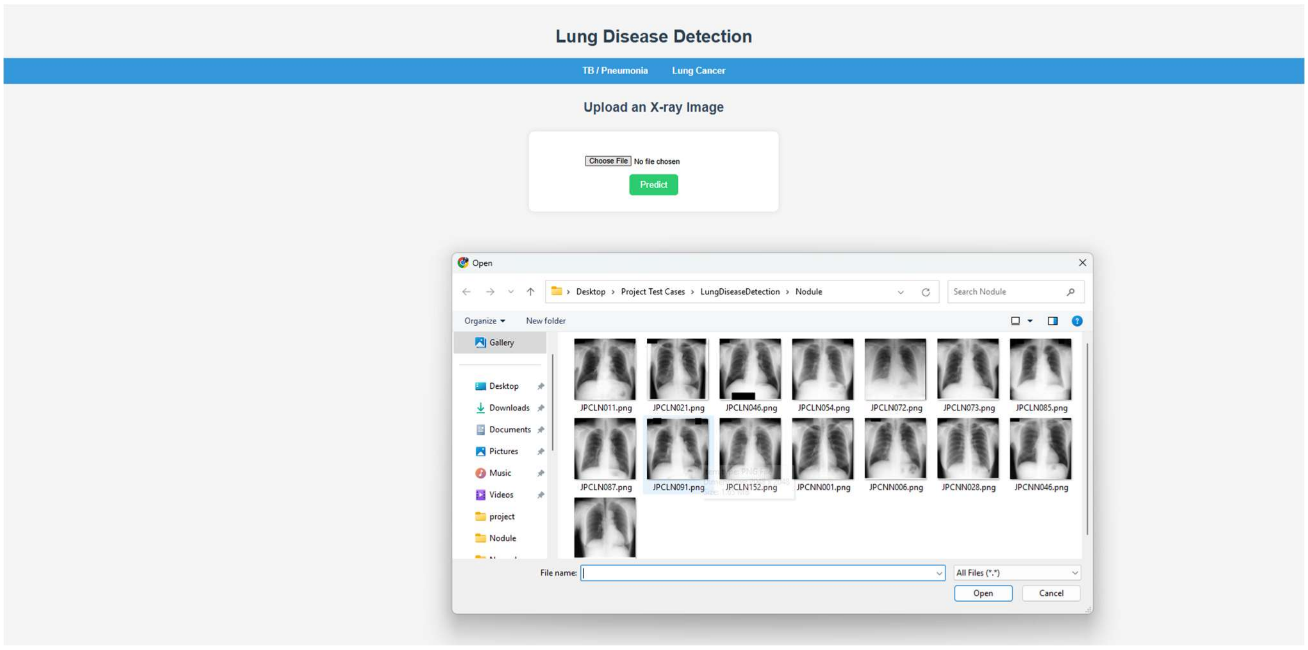


Figure 6.2: Giving user input through interface

`

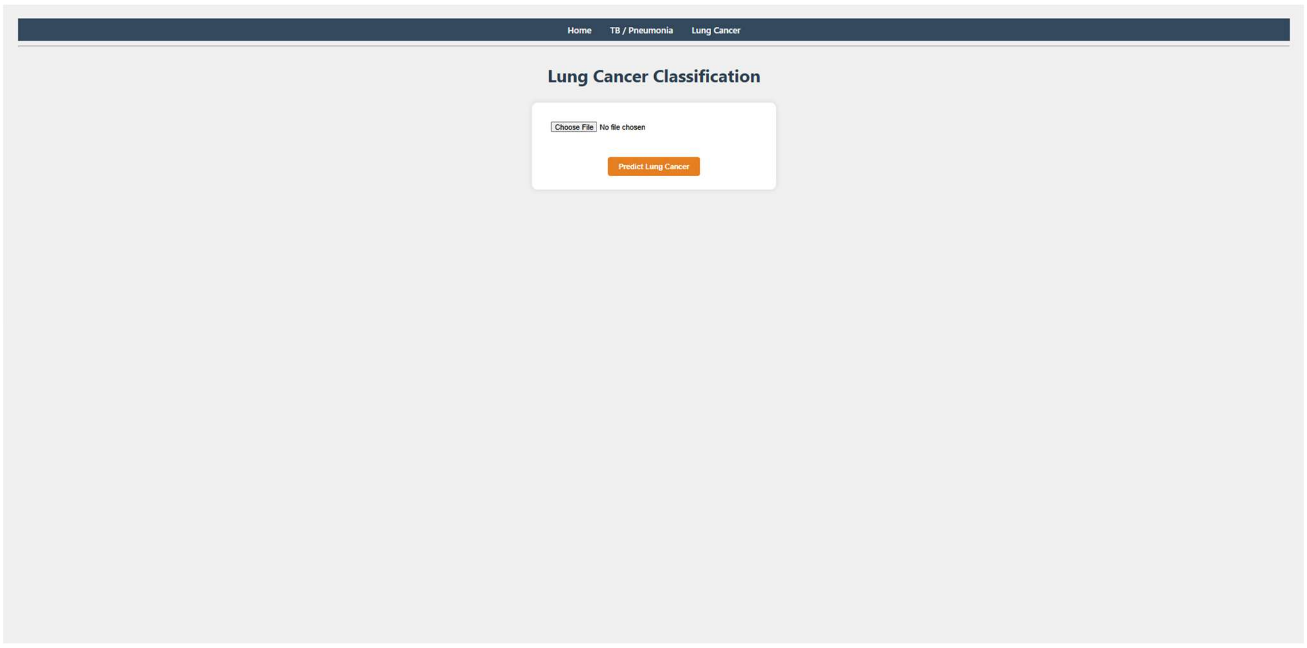


Figure 6.3: Individual lung cancer classification model

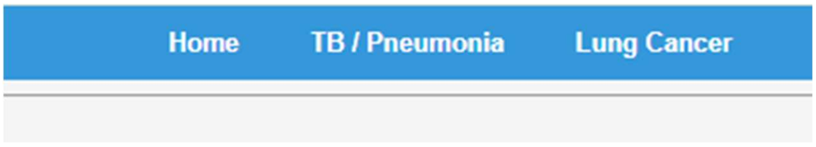


Figure 6.4: Navigation bar

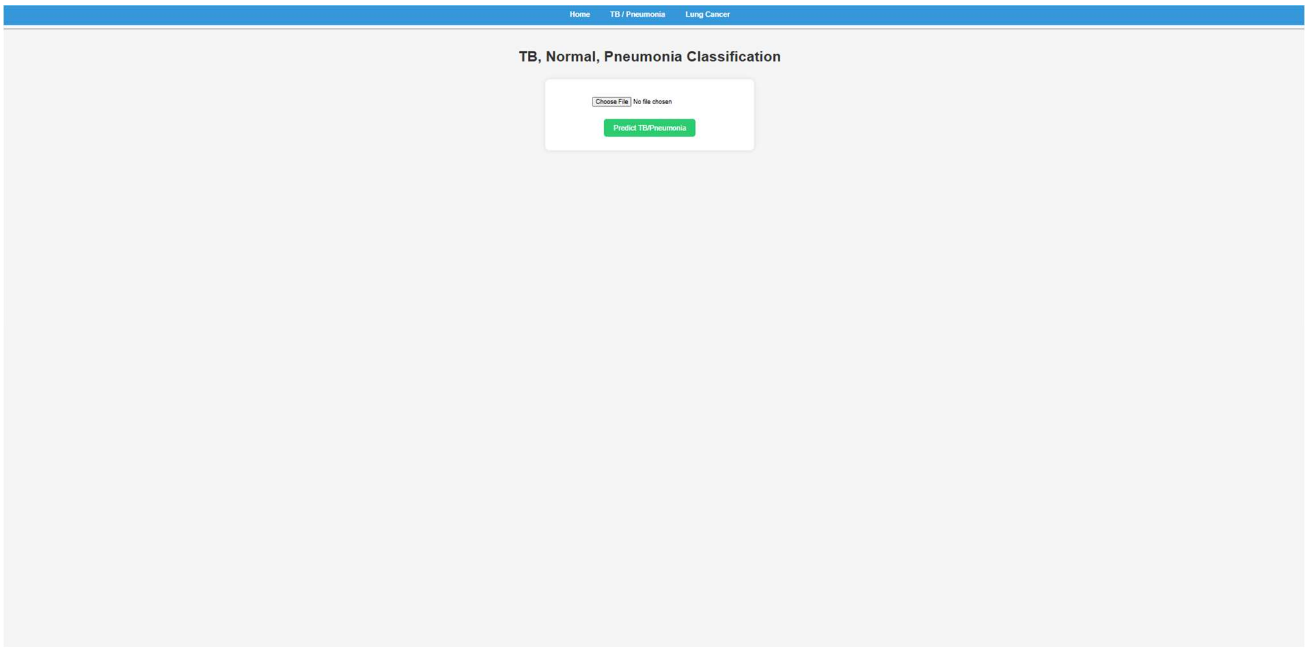
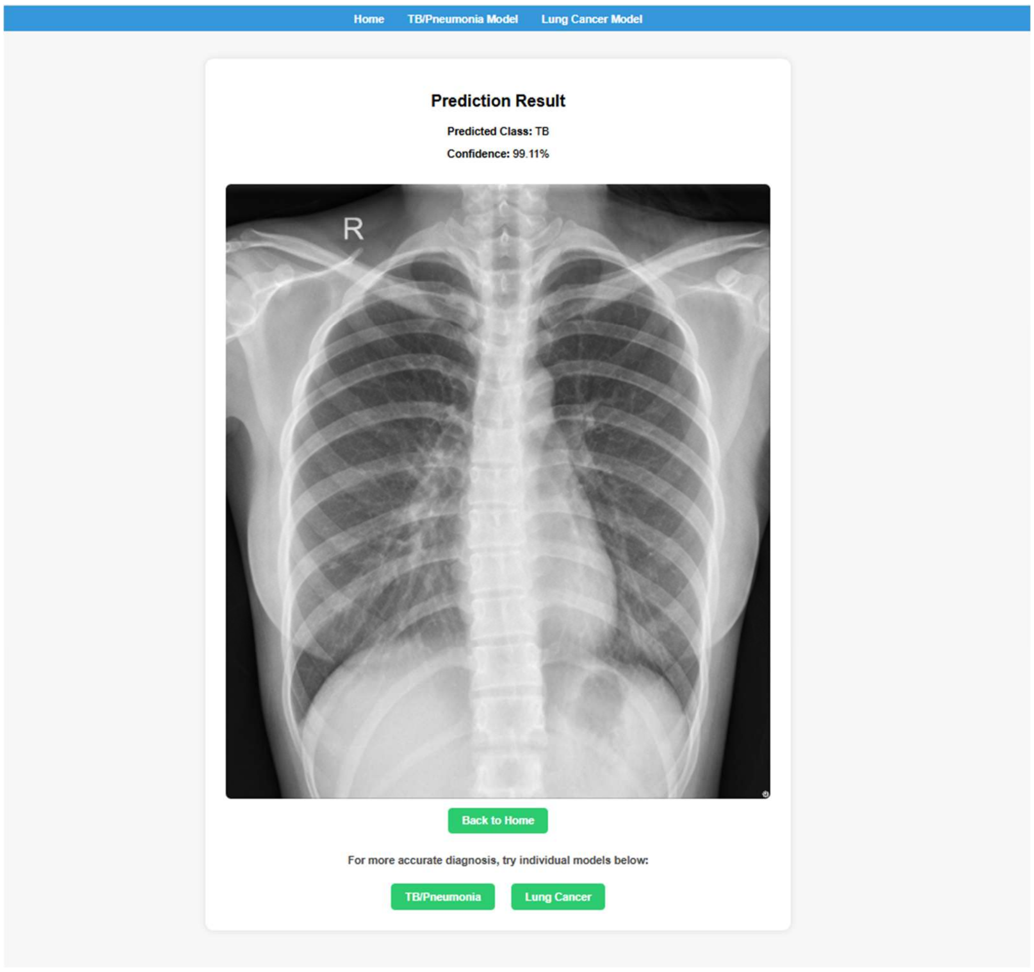


Figure 6.5: Individual Tb_pneumonia classification model



Prediction Result

Predicted Class: TB

Confidence: 99.11%

For more accurate diagnosis, try individual models below:

Figure 6.6: Results Page

6.3 Key Findings

The lung disease detection system, built using deep learning-based image classification, yielded several important outcomes and insights:

1. High Classification Accuracy

The convolutional neural network (CNN) model demonstrated strong performance in detecting lung conditions from X-ray images. It achieved:

- Training Accuracy: 98.91%
- Validation Accuracy: 95.45%
- Test Accuracy: 93.43%

These metrics indicate a highly effective model capable of reliably identifying diseases like Nodule, Pneumonia, TB, and Normal cases.

2. Balanced Class Performance

The model showed high precision and recall across all four classes:

- Nodule: Precision 0.97, F1-score 0.92

`

- Pneumonia: Precision 0.98, F1-score 0.96
- Normal and TB: Consistently high F1-scores (0.94 and 0.92 respectively)

The balanced confusion matrix confirms the model's generalization capability without significant class bias.

3. Lightweight and Real-Time Compatibility

The CNN model, trained on grayscale chest X-rays, has a small inference time, making it ideal for real-time disease detection in clinical or rural settings using minimal computational resources.

4. Practical Application in Low-Resource Settings

This system is highly suited for rural areas with limited access to radiologists. It aids healthcare workers by providing instant diagnostic suggestions, supporting faster patient triage and treatment.

5. Robust Performance Under Input Variability

The model was tested against variations in X-ray size, contrast, and orientation, and still maintained reliable outputs, proving robustness across real-world scenarios.

6.4 Challenges Faced

Despite the success of the system, several challenges were encountered during development:

1. Data Quality and Imbalance

Some classes like Nodule had fewer samples compared to others, causing slight variability in recall. Careful augmentation and class balancing strategies were needed to address this.

2. Dataset Diversity

Public chest X-ray datasets often vary in format and labeling. Harmonizing image sizes and cleaning labels required significant preprocessing effort.

3. Model Generalization

Although the model performed well on test data, adapting it to new hospital environments with different equipment or imaging conditions required additional validation and potential fine-tuning.

4. Visualization and Interpretability

To build trust with medical professionals, the need for visual explanations (e.g., Grad-CAM heatmaps) was evident. This added interpretability helps indicate which regions of the X-ray the model focused on.

`

5. Deployment of Large Models

The classification model sizes (some >150MB) posed challenges during deployment, especially on free platforms with limited storage or bandwidth. Techniques like model quantization or cloud-based loading had to be considered.

6. Simplicity in UI Design

Since the app is aimed at both doctors and non-technical users in rural clinics, the web interface needed to be simple, fast, and intuitive—yet still communicate complex predictions and confidence scores clearly.

`

CHAPTER 07
CONCLUSION

7.1 CONCLUSION

The proposed Lung Disease Detection System using Deep Learning offers a fast, reliable, and cost-effective diagnostic tool for identifying Tuberculosis (TB), Pneumonia, Pulmonary Nodules, and Normal lungs from chest X-ray images. By leveraging Convolutional Neural Networks (CNNs), the system achieves high accuracy and robustness in classifying lung conditions based on medical imaging.

With a training accuracy of 98.91%, validation accuracy of 95.45%, and test accuracy of 93.43%, the model demonstrates strong generalization capabilities across diverse chest X-ray samples. High precision, recall, and F1-scores across all classes further validate its diagnostic effectiveness.

The web-based interface built using Flask makes it highly accessible for use in rural or under-resourced healthcare settings, providing a second-opinion system that can assist healthcare professionals and reduce misdiagnosis. The lightweight design allows for smooth deployment on low-resource servers and ensures fast inference.

This project showcases the practical application of AI in medical diagnostics, particularly for respiratory diseases, and paves the way for broader integration of machine learning tools into public healthcare infrastructure. The system has the potential to reduce dependency on expert radiologists, improve early detection rates, and save lives by facilitating quicker medical intervention.

7.2 FUTURE WORK

While the current implementation delivers strong results, several enhancements can be made to expand its capabilities and real-world applicability:

1. Dataset Expansion

- Include a more diverse set of X-rays from various demographics and imaging devices to improve generalization.
- Expand to other lung diseases such as COVID-19, Fibrosis, or Lung Cancer.

2. Multi-label Classification

- Enable the model to handle multiple simultaneous conditions (e.g., TB and Nodule in one scan) using multi-label classification.

`

3. Grad-CAM Integration

- Integrate Grad-CAM (Gradient-weighted Class Activation Mapping) to highlight infected or abnormal regions on the X-ray. This will increase model interpretability for clinicians.

4. Model Compression & Optimization

- Implement model quantization or pruning to reduce model size and enable smooth deployment on mobile and edge devices.

5. Telemedicine & Mobile Integration

- Develop a mobile application for remote diagnostics and X-ray uploads, particularly useful in rural and remote areas.
- Integrate with telemedicine platforms to support virtual consultations and diagnostics.

6. Clinical Validation

- Collaborate with medical institutions to conduct clinical trials, gather feedback from radiologists, and validate performance in real-world settings.

7. Cloud or API-based Deployment

- Provide an API endpoint or cloud-hosted version to make the model easily accessible across hospitals and health centers without requiring local deployment.

`

REFERENCES

1. Gulshan, V., et al. (2016). *Development and Validation of a Deep Learning Algorithm for Detection of Diabetic Retinopathy in Retinal Fundus Photographs*. JAMA, 316(22), 2402–2410. https://doi.org/10.1001/jama.2016.17216
2. Pratt, H., & Seneviratne, S. (2016). *Deep Learning for Medical Image Analysis: A Review*. Journal of Biomedical Informatics, 60, 106–121. https://doi.org/10.1016/j.jbi.2016.03.003
3. Kermany, D. S., et al. (2018). *Identifying Medical Diagnoses and Treatable Diseases by Image-Based Deep Learning*. Cell, 172(5), 1122–1131.e9. https://doi.org/10.1016/j.cell.2018.02.010
4. Redmon, J., et al. (2016). *You Only Look Once: Unified, Real-Time Object Detection*. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 779–788. https://doi.org/10.1109/CVPR.2016.91
5. Rajpurkar, P., et al. (2017). *CheXNet: Radiologist-Level Pneumonia Detection on Chest X-Rays with Deep Learning*. Stanford ML Group. arXiv preprint arXiv:1711.05225. https://arxiv.org/abs/1711.05225
6. Srinivasan, A., et al. (2021). *Lung Disease Detection from Chest X-ray Images Using Convolutional Neural Networks*. IEEE Access, 9, 50461–50472. https://doi.org/10.1109/ACCESS.2021.3068653
7. Wang, X., Peng, Y., Lu, L., et al. (2017). *ChestX-ray8: Hospital-scale Chest X-ray Database and Benchmarks on Weakly-Supervised Classification and Localization of Common Thorax Diseases*. In *CVPR*. https://doi.org/10.1109/CVPR.2017.369
8. Bar, Y., et al. (2015). *Chest Pathology Detection Using Deep Learning with Non-medical Training*. In *2015 IEEE 12th International Symposium on Biomedical Imaging (ISBI)*, 294–297. https://doi.org/10.1109/ISBI.2015.7163871
9. Zhang, Z., et al. (2020). *A Deep Learning Framework for Recognizing Pneumonia from Chest X-ray Images*. Healthcare, 8(4), 501. https://doi.org/10.3390/healthcare8040501
10. Cohen, J. P., et al. (2020). *COVID-19 Image Data Collection*. arXiv preprint arXiv:2003.11597. https://arxiv.org/abs/2003.11597